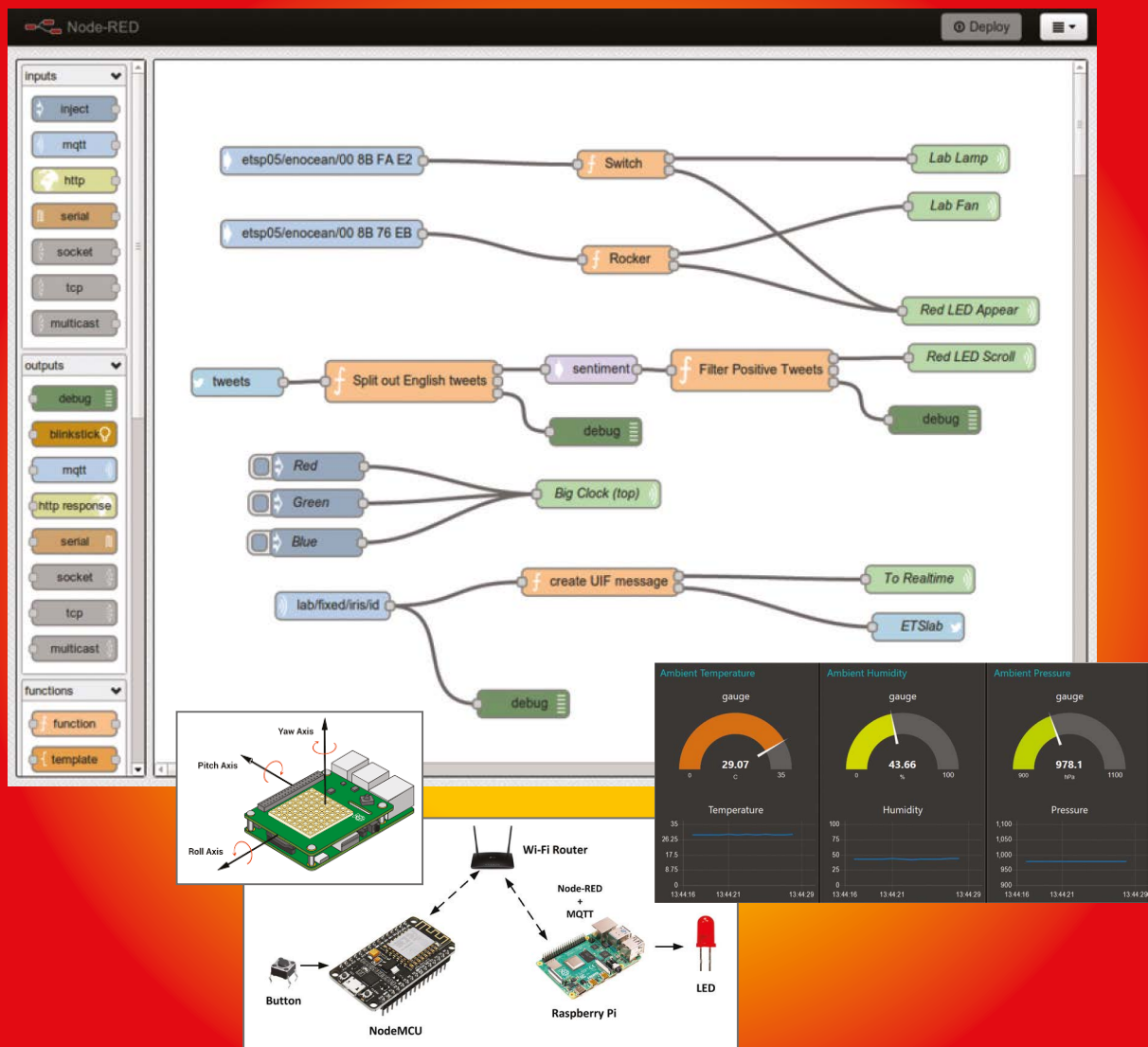


Design IoT Projects with Raspberry Pi, Arduino and ESP32

Programming with Node-RED



Programming with Node-RED

Design IoT Projects with Raspberry Pi, Arduino and ESP32



Dogan Ibrahim



elektor

LEARN > DESIGN > SHARE

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street, London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

● All rights reserved. No part of this book may be reproduced in any material form, including photocopying, or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication, without the written permission of the copyright holder except in accordance with the provisions of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd., 90 Tottenham Court Road, London, England W1P 9HE. Applications for the copyright holder's permission to reproduce any part of the publication should be addressed to the publishers.

● Declaration

The author and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, or hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause..

● British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

● **ISBN 978-1-907920-88-2**

© Copyright 2020: Elektor International Media b.v.

Prepress Production: D-Vision, Julian van den Berg

First published in the United Kingdom 2020



Elektor is part of EIM, the world's leading source of essential technical information and electronics products for pro engineers, electronics designers, and the companies seeking to engage them. Each day, our international team develops and delivers high-quality content - via a variety of media channels (including magazines, video, digital media, and social media) in several languages - relating to electronics design and DIY electronics. www.elektormagazine.com

LEARN > DESIGN > SHARE

Preface	13
Chapter 1 • Raspberry Pi 4	15
1.1 Overview	15
1.2 Parts of the Raspberry Pi 4	15
1.3.1 Setup option 1	18
1.3.2 Setup option 2	19
Summary	19
Chapter 2 • Installing the Raspberry Pi operating system	20
2.1 Overview	20
2.2 Raspbian Buster installation steps on Raspberry Pi 4	20
2.3 Remote access	22
2.4 Using Putty	23
2.4.1 Configuring Putty	24
2.5 Remote access of the desktop	25
2.6 Summary	26
Chapter 3 • Using the command line	27
3.1 Overview	27
3.2 The Raspberry Pi directory structure	27
3.3 File permissions	28
3.4 Help	32
3.5 Date, time, and calendar	33
3.6 File operations	33
3.7 System and user information	35
3.8 Resource monitoring on Raspberry Pi	37
3.9 Shutting down	39
3.10 Summary	40
Chapter 4 • Installing Node-RED software on Raspberry Pi	41
4.1 Overview	41
4.2 Raspberry Pi Node-RED installation	41
4.3 Node-RED interface to external world	43
4.4 Node-RED screen layout	44
4.5 Project 1 – Hello World!	44

4.6 Project 2 – Date and time	47
4.7 Project 3 – Temperature conversion	52
4.8 Importing and exporting flow programs	54
4.8.1 Exporting flows.	54
4.8.2 Importing aflow	55
4.9 Copying nodes in the same workspace	55
4.10 Core nodes	55
4.10.1 Input nodes	55
4.10.2 Output nodes	56
4.10.3 Function nodes	57
4.10.4 Social nodes	58
4.10.5 Storage nodes	58
4.10.6 Analysis nodes	58
4.10.7 Advanced nodes	58
4.10.8 Raspberry Pi nodes	58
4.11 Project 4 – Dice number	59
4.12 Project 5 – Double dice numbers	60
4.13 Project 6 – Unit conversions - Multiple inputs to a function	62
4.14 Multiple inputs and multiple outputs from a function	65
4.15 Project 7 – Mean of numbers - Using the smooth node.	67
4.16 Project 8 – Squares of numbers.	68
4.17 Project 9 – Getting the weather reports – Display the local weather report.	70
4.18 Project 10 – Display the current temperature	73
4.19 Project 11 – Sending the current temperature to an email	74
4.20 Project 12 – Sending the current temperature and atmospheric pressure to your twitter account	76
4.21 Node-RED configuration	78
4.22 Summary	79
Chapter 5 • Node-RED based Raspberry Pi projects using GPIO.	80
5.1 Overview	80
5.2 GPIO – Parallel interface.	80
5.3 Project 13 – LED control.	82
5.4 Project 14 – Flashing LED	85

5.4.1 Using a context variable to flash the LED	87
5.5 Project 15 – Alternately flashing LEDs	88
5.5.1 Using context variables to flash the LEDs alternately.	89
5.6 Project 16 – Temperature alarm with buzzer.	90
5.7 Project 17 – Controlling a GPIO port remotely using email	93
5.8 Project 18 – Confirmation of the buzzer state	95
5.9 Project 19 – Controlling multiple GPIO ports remotely using email.	98
5.10 Project 20 – Traffic lights simulator	102
5.11 Project 21 – Push-Button switch input	106
5.12 Project 22 – Changing LED brightness – The PWM output.	110
5.13 Summary	112
6.1 Overview	113
6.2 Raspberry Pi I ² C ports	114
6.3 I ² C LCD	115
6.4 Installing the I ² C LCD software on Node-RED	115
6.5 Project 23 – Displaying current time on the LCD	118
6.6 Project 24 – Displaying local temperature and humidity on the LCD.	120
6.7 Project 25 – Displaying dice numbers.	120
6.8 Project 26 – Event counter with LCD	122
6.9 Project 27 – DHT11 temperature and humidity sensor with LCD Display.	124
6.10 Project 28 – Ultrasonic distance sensor with LCD display	129
6.11 Project 29 – Ultrasonic distance alarm with buzzer	134
6.12 Project 30 – Ultrasonic car parking system with buzzer	135
6.13 Using a parallel LCD.	137
6.14 Project 31 – Displaying message on a parallel LCD	139
6.15 Summary	141
Chapter 7 • Using ADC in Raspberry Pi Node-RED projects.	142
7.1 Overview	142
7.2 The MCP3002 ADC.	142
7.3 Project 32 – Voltmeter with LCD output	144
7.4 Project 33 – Temperature measurement using analog sensor	146
7.5 Project 34 – ON/OFF temperature control.	148

7.6 Summary	152
Chapter 8 • The dashboard palette	153
8.1 Overview	153
8.2 Installing the dashboard.	153
8.3 Project 35 – Using a gauge to display the temperature	154
8.4 Using a Line Chart to display the temperature.	156
8.5 Using a Bar Chart to display the temperature	157
8.6 Project 36 – Using gauges to display the temperature and humidity	158
8.7 Project 37 – Using multiple gauges	160
8.8 Project 38 – Using a slider to change LED brightness.	161
8.9 Project 39 – Using button nodes to control an LED	163
8.10 Project 40 – Using switch and text nodes to control an LED	165
8.11 Project 41 – Talking weather forecast.	166
8.12 Configuring the dashboard	169
8.13 Summary	169
Chapter 9 • Wi-Fi UDP/TCP Network-Based projects.	170
9.1 Overview	170
9.2 Project 42 – Controlling an LED from a mobile phone – UDP based communication	170
9.3 Project 43 – Controlling multiple LEDs from a mobile phone – UDP based communication	173
9.5 Project 45 – Controlling an LED from a mobile phone – TCP based communication	178
9.6 Project 46 – Controlling Multiple LEDs From a Mobile Phone – TCP Based Communication	181
9.7 Project 47 – Sending the temperature and humidity to the mobile phone – TCP based communication	182
9.8 Project 48 – Chat program – Mobile phone to Raspberry Pi chat	184
9.9 Project 49 – Using the ping.	187
9.10 Summary	189
Chapter 10 • Storage Nodes.	190
10.1 Overview	190
10.2 Project 50 – Storing timestamped temperature and humidity data in a file	190

10.3 Project 51 – Reading contents of a file	192
10.4 Project 52 – Reading a file line by line	195
10.5 Summary	197
Chapter 11 • Serial communication	198
11.1 Overview	198
11.2 Project 53 – Communicating with Arduino over serial line	199
11.3 Project 54 – Receiving serial data – Receiving GPS data	204
11.4 Project 55 – Receiving GPS data – Extracting the latitude and longitude.	208
11.5 Project 56 – Displaying our location on a map	212
11.6 Project 57 – Sending out serial data to arduino	216
11.7 Project 58 – Connecting Raspberry Pi and Arduino Uno using USB ports.	220
11.8 Project 59 – Sending serial data from Raspberry Pi to Arduino over RF Radio . . .	222
11.9 Summary	224
Chapter 12 • Using Sense HAT	225
12.1 Overview	225
12.2 The Sense HAT board	225
12.3 Node-RED Sense HAT nodes	226
12.4 Project 60 - Displaying the temperature, humidity, and pressure (environmental events)	227
12.5 Project 61 - Displaying the compass heading (motion events)	229
12.6 Project 62 - Displaying the acceleration (motion events)	230
12.7 Project 63 - Displaying the orientation (motion events)	231
12.8 Using the joystick	233
12.9 Using the LED matrix	234
12.10 Project 64 – Random flashing LED lights having random colours	238
12.11 Project 65 – Display of temperature by LED count	239
12.12 Displaying and scrolling data on LED Matrix	242
12.13 Project 66 – Scrolling the pressure readings on the LED matrix	242
12.14 Summary	244
Chapter 13 • Node-RED with Arduino Uno	245
13.1 Overview	245
13.2 Installing Node-RED For Arduino Uno	245

13.3 Project 67 – Flashing LED	247
13.4 Project 68 – Displaying the ambient temperature in the debug window	248
13.5 Project 69 – Displaying the ambient temperature in the dashboard	250
13.6 Project 70 – Displaying the ambient temperature as gauge and chart	251
13.7 Using the Arduino Uno serial port	253
13.7.1 Project 71 – Using the DHT11 with the Arduino	253
13.8 Summary	256
Chapter 14 • Using the ESP32 DevKitC with Node-RED	257
14.1 Overview	257
14.2 ESP32 DevKitC and Node-RED.	259
14.3 Project 72 – Controlling an LED connected to ESP32 DevKitC	259
14.4 Summary	262
Chapter 15 • Using Amazon Alexa in Node-RED projects	263
15.1 Overview	263
15.2 Project 73 – Controlling an LED using Alexa	263
15.3 Project 74 – Controlling an LED and a buzzer using Alexa.	265
15.4 Summary	268
Chapter 16 • Accessing Raspberry Pi Node-RED from anywhere	269
16.1 Overview	269
16.2 The ngrok	269
16.3 Starting Node-RED automatically at reboot time	271
16.4 Summary	271
Chapter 17 • Using Bluetooth with Node-RED	272
17.1 Overview	272
17.2 Project 75 – Controlling an LED and a buzzer using Bluetooth.	272
17.3 Summary	276
Chapter 18 • Node-RED and MQTT	278
18.1 Overview	278
18.2 How MQTT works.	278
18.3 The Mosquitto broker	280
18.4 Using MQTT in home automation and in IoT projects	281
18.5 Project 76 – Controlling an LED using MQTT	282

18.6 The ESP8266 processor	283
18.7 Project 77 – Flashing an LED using ESP8266 NodeMCU	285
18.8 Using the ESP8266 NodeMCU with MQTT	287
18.9 Project 78 – Controlling an LED using ESP8266 NodeMCU with MQTT – LED connected to Raspberry Pi	288
18.10 Project 79 – Controlling an LED using ESP8266 NodeMCU with MQTT – LED connected to ESP8266 NodeMCU	293
18.11 Summary	299
Chapter 19 • Using HTTP in Node-RED projects.	300
19.1 Overview	300
19.2 Using HTTP GET.	300
19.3 Web server	301
19.4 Project 80 – Controlling 4 Relays using web server	302
19.5 Summary	308
Appendix A • The function node.	309
A.1 Overview	309
A.2 Variables	309
A.3 Multiple outputs	310
A.4 String manipulation	310
A.5 Mathematical functions	313
A.6 Number conversions and checking numbers	314
A.7 Date	315
A.8 Arrays	315
A.9 Conditional statements	316
A.10 Repetition (loops)	317
A.12 Examples	319
Appendix B • Flow programs used in the book	321
A.1 Overview	321
A.2 Using the flow programs	321
Appendix C • Components used in the book	322
Index	323

Preface

It is becoming important for microcontroller users to adapt to new technologies quickly and learn the architecture and use of high-performance 32-bit microcontrollers. Several manufacturers offer 32-bit microcontrollers as general-purpose processors in embedded applications. ARM architecture is currently one of the most frequently used 32-bit microcontroller architectures in mobile devices, such as in mobile phones, iPads, games consoles, and in many other portable devices.

Raspberry Pi is based on ARM architecture and it is currently one of the most commonly used single-board computers used by students, engineers, and hobbyists. There are hundreds of Raspberry Pi based projects available on the internet

Arduino is also a very popular microcontroller development board. Although it is based on 8-bit architecture, it is widely used as it is supported by large numbers of software libraries, making it easy to develop projects in relatively short times.

Another popular microcontroller is ESP32. It is widely sold as a development board known as the ESP32 DevKitC. The reason why ESP32 is very popular is that it has onboard Wi-Fi and Bluetooth capability, many digital and analog input ports, and built-in timers. Additionally, its power consumption is very low and it has a processor that can be put into sleep mode, which consumes extremely low current.

The Internet of Things (IoT) is becoming a major application area of embedded systems. As a result, more people are becoming interested in learning about embedded design and programming. In addition, we can see that more technical colleges and universities are moving away from legacy 8-bit and 16-bit microcontrollers and introducing 32-bit embedded microcontrollers into their curriculums. Some IoT applications demand precision, high processing power, and very low power consumption.

Node-RED is an open-source visual editor for wiring the Internet of Things produced by IBM. Node-RED comes with a large number of nodes to handle a variety of tasks. The required nodes are selected and joined together to perform a particular task. Node-RED is based on flow type programming where nodes are configured and joined together to form an application program. There are nodes for doing very complex tasks, including web access, Twitter, E-mail, HTTP, Bluetooth, MQTT, controlling GPIO ports, etc. The nice thing about Node-RED is that the programmer does not need to learn how to write complex programs. For example, an email can be sent by joining a few nodes together and writing a few lines of code.

The aim of this book is to teach how Node-RED can be used in projects. The main processor used in the majority of projects in this book is Raspberry Pi 4. Chapters are included to show how Node-RED can be used with Arduino Uno, ESP32 DevKitC, and the ESP8266 NodeMCU microcontroller development boards.

Many example projects are given in the book. All projects have been fully tested and were working at the time of writing this book. Users can select flow programs of the projects from the book website and configure them to suit their own applications. The operation of each flow program is fully described in the book.

I hope you enjoy reading this book and are then able to use Node-RED happily in your future projects.

Prof Dr. Dogan Ibrahim
London, 2020

Chapter 1 • Raspberry Pi 4

1.1 Overview

Raspberry Pi has recently become one of the most popular and most powerful single-board computers used by students, hobbyists, and professional engineers. Raspberry Pi 4 is the latest and the most powerful version of Raspberry Pi. In this chapter, we will be looking at the basic specifications and requirements of the Raspberry Pi 4 computer. What is included in this chapter can easily be applied to other models in the Raspberry Pi family.

1.2 Parts of the Raspberry Pi 4

Just like its earlier versions, Raspberry Pi 4 is a single-board computer having the following basic specifications:

- 1.5GHz 64-bit quad-core CPU
- 1GB, 2GB, or 4GB RAM
- 2.4GHz and 5.0GHz IEEE 802.11ac Wi-Fi
- Bluetooth 5.0 BLE
- Gigabit Ethernet
- 2 x USB 3.0, 2 x USB 2.0 and 1 x USB-C ports
- 2 x micro-HDMI ports for dual display, supporting up to 4K resolution
- DSI display and CSI camera ports
- micro SD card slot for the operating system and data storage
- 4-pole stereo audio and composite video port
- 40-pin GPIO header
- Power over Ethernet (PoE) enabled with the PoE HAT
- OpenGL ES 3.0 graphics

Figure 1 shows the Raspberry Pi 4 board with its major components identified.

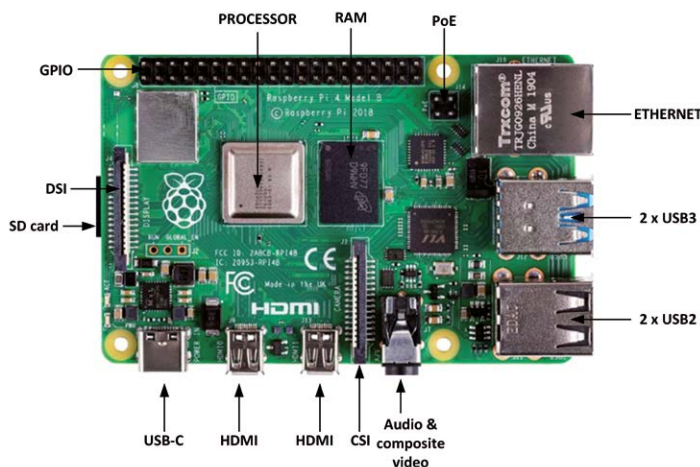


Figure 1.1 Raspberry Pi 4 board

A brief description of the various components on the board is given below:

Processor: the processor is enclosed in a metal cap and it is based on Broadcom BCM2711B0, which consists of a Cortex A-72 core, operating at 1.5GHz.

RAM: There are 3 versions of Raspberry Pi 4 depending on the amount of DDR4 RAM required: 1GB, 2GB, and 4GB.

USB Ports: Raspberry Pi 4 includes 2 x USB 3.0, 2 x USB 2.0, and 1 x USB-C ports. USB 3.0 data transfer rate is 4,800 Mbps (megabits per second), while USB 2.0 can transfer at up to 480Mbps, i.e. 10 times slower than the USB 2.0. The USB-C port enables the board to be connected to a suitable power source.

Ethernet: The Ethernet port enables the board to be connected directly to an Ethernet port on a router. The port supports Gigabit connections (125Mbps).

HDMI: Two micro HDMI ports are provided that support up to 4K screen resolutions. HDMI adapters can be used to interface the board to standard size HDMI devices.

GPIO: A 40-pin header is provided as the GPIO (General Purpose Input Output). This is compatible with the earlier GPIO ports.

Audio and Video Port: A 3.5mm jack type socket is provided for stereo audio and composite video interface. Headphones can be connected to this port. External amplifier devices will be required to connect speakers to this port. This port also supports composite video, enabling TVs, projectors, and other composite video compatible display devices to be connected to the port.

CSI Port: This is the camera port (Camera Serial Interface), allowing a compatible camera to be connected to the Raspberry Pi.

DSI Port: This is the display port (Display Serial Interface), allowing a compatible display (e.g. 7 inch Raspberry Pi display) to be connected to the Raspberry Pi.

PoE Port: This is a 4-pin header, allowing the Raspberry Pi to receive power from a network connection.

Micro SD Card: This card is mounted at the cardholder placed at the bottom of the board and it holds the operating system software as well as the operating system and user data. Requirements of the Raspberry Pi 4

As listed below, a number of external devices are required before the Raspberry Pi can be used:

- Power supply
- Micro SD card

- Operating system software
- USB keyboard and mouse
- A micro HDMI cable to receive sound and video signals
- HDMI compatible display or TV (you may also need to have micro HDMI to DVI-D or VGA adapters. A 3.5mm TRRS type cable and plug will be required if you will be using an old TV with composite video)

Power Supply: A 5V 3A power supply with a USB-C type connector is required. You may either purchase the official Raspberry Pi 4 power supply (Figure 1.2) or use a USB-C adapter to provide power from an external source.



Figure 1.2 Official Raspberry Pi 4 power supply

Micro SD Card: It is recommended to use a micro SD card with a capacity of at least 8GB, although higher capacity (e.g. 16GB or 32GB) is better as there will be room to grow in the future. A Class 10 (or faster) card is recommended.

Operating System: You can purchase the operating system pre-loaded on a micro SD card, known as NOOBS (New Out Of Box Software) which requires minimum configuration before it is fully functional. The alternative is to purchase a blank micro SD card and upload the operating system on this card. The steps to prepare a new micro SD card with the operating system is given in the next Chapter.

USB Keyboard and Mouse: You can either use a wireless or wired keyboard and mouse pair. If using a wired pair, you should connect the keyboard to one of the USB ports and the mouse to another USB port. If using a wireless keyboard and mouse, you should connect the wireless dongle to one of the USB ports.

Display: A standard HDMI compatible display monitor with a micro HDMI to standard HDMI adapter can be used. Alternatively, a VGA type display monitor with a micro HDMI to VGA adapter or DVI-D adapter can be used. If you have an old TV with a composite video interface, then you can connect it to the Raspberry Pi 3.5mm port with a TRRS type connector. You may also consider purchasing additional parts, such as a case, CPU fan, and so on. The case is very useful as it protects your Raspberry Pi electronics. The working temperature of the CPU can go as high as 80 degrees Centigrade. Using a fan (see Figure 1.3) makes the CPU more efficient as it can lower its temperature by about 50%.



Figure 1.3 Raspberry Pi 4 CPU fan (www.seeedstudio.com)

1.3.1 Setup option 1

As shown in Figure 1.4, in this option various devices are connected directly to the Raspberry Pi 4. Depending on what type of display monitor we have, we can use an HDMI display, VGA monitor, DVI-D monitor, or TV. Notice that depending on the external USB devices used, you can use either the USB 2.0 or the USB 3.0 ports.

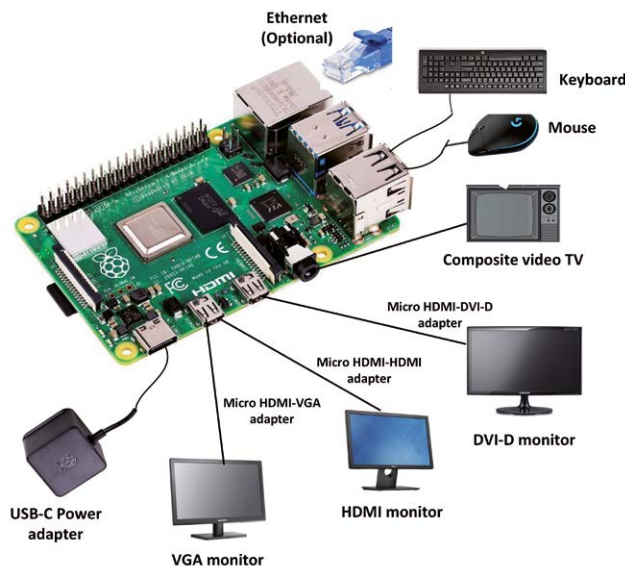


Figure 1.4 Raspberry Pi 4 setup - option 1

1.3.2 Setup option 2

In this option, shown in Figure 1.5, a powered hub is used to connect the USB devices.

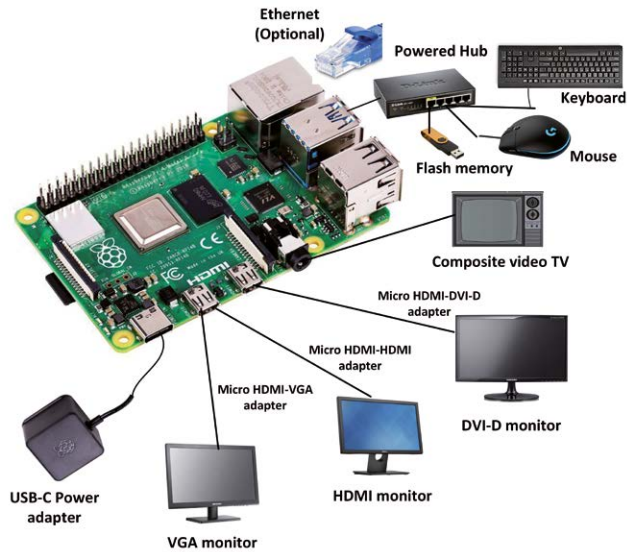


Figure 1.5 Raspberry Pi 4 setup – option 2

Summary

In this chapter, we have learned about the main parts and their functions of the Raspberry Pi 4 board. In addition, we have learned how to configure Raspberry Pi 4.

In the next chapter, we will be learning how to install the latest Raspberry Pi operating system, Raspbian Buster, on a new blank micro SD card.

Chapter 2 • Installing the Raspberry Pi operating system

2.1 Overview

In the last chapter, we had a look at some of the hardware features of Raspberry Pi 4 and learned how to set up the hardware using various external devices. In this chapter, we will be learning how to install the latest Raspberry Pi operating system, Raspbian Buster, on an SD card.

2.2 Raspbian Buster installation steps on Raspberry Pi 4

Raspbian Buster is the latest operating system of Raspberry Pi 4. This section gives the steps for installing this operating system on a new blank SD card, ready to use with your Raspberry Pi 4. You will need a micro SD card with a capacity of at least 8GB (16 GB is preferable) before installing the new operating system on it.

The steps to install Raspbian Buster are as follows:

- Download the Buster image to a folder on your PC (e.g. C:\RPiBuster) from the following link by clicking the Download ZIP under section **Raspbian Buster with desktop and recommended software** (see Figure 2.1). At the time of writing this book the file was called: **2019-07-10-raspbian-buster-full.img**. You may have to use the Windows 7Zip software to unzip the download since some of the features are not supported by older unzip software.

<https://www.raspberrypi.org/downloads/raspbian/>

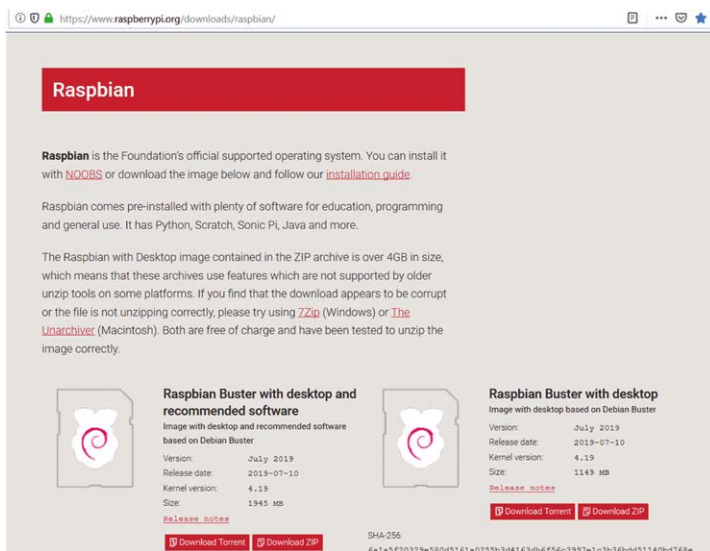


Figure 2.1 Raspbian Buster download page

- Put your blank micro SD card into the card slot on your computer. You may need to use an adapter to do this.

- Download the Etcher program to your PC to flash the disk image. The link is (see Figure 2.2):

<https://www.balena.io/etcher/>

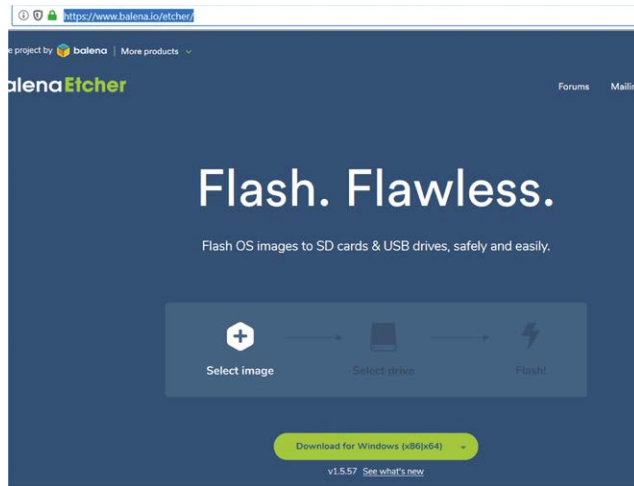


Figure 2.2 Download Etcher

- Double click to Open Etcher, and click **Select image**. Select the Raspbian Buster file you just downloaded.
- Click **Select target** and select the micro SD card.
- Click **Flash** (see Figure 2.3). This may take several minutes, wait until it is finished. The program will then validate and unmount the micro SD card. You can remove your micro SD card after it is unmounted.

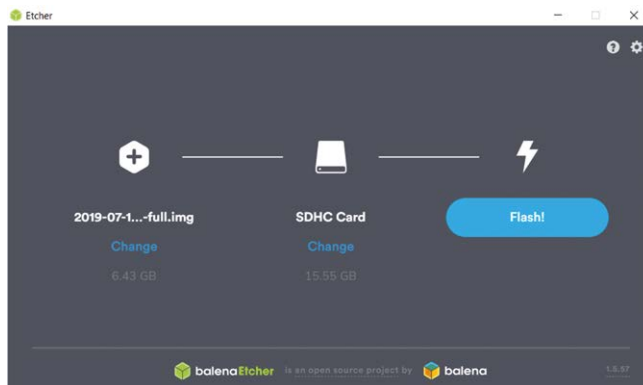


Figure 2.3 Click Flash to flash the disk image

- You are now ready to use your micro SD card on your Raspberry Pi 4.

- Connect your Raspberry Pi 4 to a HDMI monitor (you may need to use an adapter cable for mini HDMI to standard HDMI conversion), connect a USB keyboard, and power up the Raspberry Pi.
- You will see the startup menu displayed on the monitor. Click Next to get started.
- Select the Wi-Fi network and enter the password of your Wi-Fi router.
- Click on the Wi-Fi icon at the top right-hand side of the screen and note the Wireless IP address of your Raspberry Pi (notice that this IP address is not static and it can change next time you power-up your Raspberry Pi).
- You should now be ready to use your Raspberry Pi 4 (see Desktop in Figure 2.4).

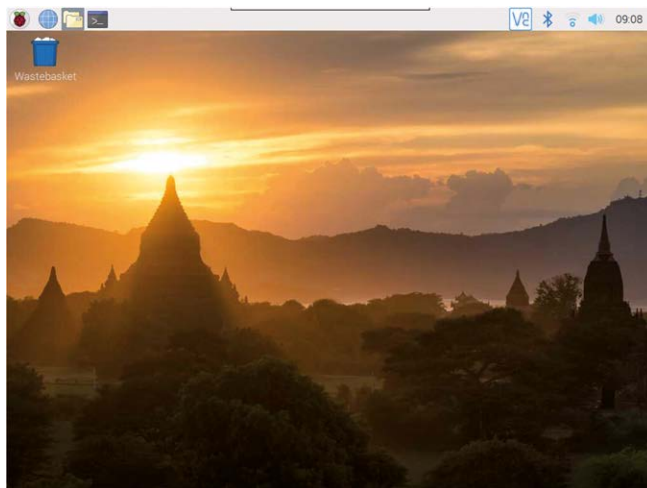


Figure 2.4 Raspberry Pi 4 desktop

Notice that the IP address of your Raspberry Pi can also be seen in your router. You can also get the IP address of your Raspberry Pi using your mobile phone. There are several programs free of charge that can be installed on your mobile phone that will show you the IP addresses of all the devices connected to your router. In this section, the use of the Android **apps** called **Who's On My Wi-Fi – Network Scanner** by *Magdalm* is used to show how the IP address of your Raspberry Pi can be displayed. Running this program will display the Raspberry Pi Wireless IP address under the heading **Raspberry Pi Trading Ltd**. In addition to the IP address, other parameters such as MAC address, gateway address, IP mask, etc are all displayed by this program.

2.3 Remote access

It is much easier to access the Raspberry Pi remotely over the Internet, for example using a PC rather than connecting a keyboard, mouse, and display to it. Before being able to access your Raspberry Pi remotely, we have to enable the SSH and the VNC by entering the following command at a terminal session:

```
pi$raspberrypi:~ $ sudo raspi-config
```

Go to the configuration menu and select **Interface Options**. Go down to **P2 SSH** (see Figure 2.5) and enable SSH. Click **<Finish>** to exit the menu.

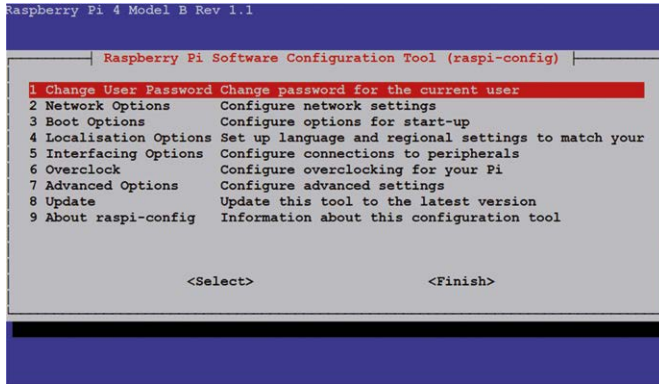


Figure 2.5 Enable SSH

You should also enable VNC so that your Raspberry Pi can be accessed graphically over the Internet. This can be done by entering the following command at a terminal session:

```
pi$raspberrypi:~ $ sudo raspi-config
```

Go to the configuration menu and select **Interface Options**. Go down to **P3 VNC** and enable VNC. Click **<Finish>** to exit the menu. At this stage, you may want to shut down your Raspberry Pi by clicking the **Applications Menu** on Desktop and selecting the **Shutdown** option.

2.4 Using Putty

Putty is a communications program that is used to create a connection between your PC and Raspberry Pi. This connection uses a secure protocol called SSH (Secure Shell). Putty doesn't need to be installed as it can just be stored in any folder of your choice and run from there.

Putty can be downloaded from the following web site:

<https://www.putty.org/>

Simply double click to run it and the Putty startup screen will be displayed. Click **SSH** and enter the Raspberry Pi IP address, then click **Open** (see Figure 2.6). The message shown in Figure 2.7 will be displayed the first time you access your Raspberry Pi. Click **Yes** to accept this security alert.

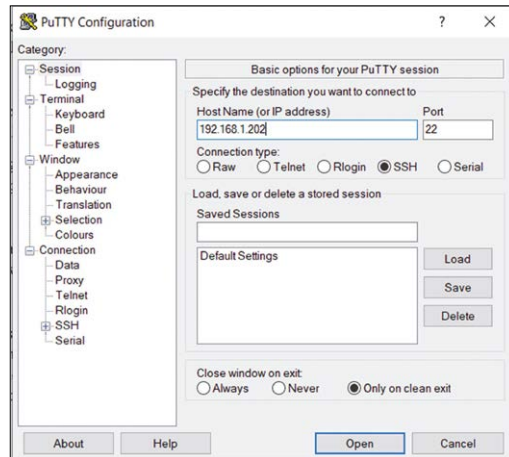


Figure 2.6 Putty startup screen

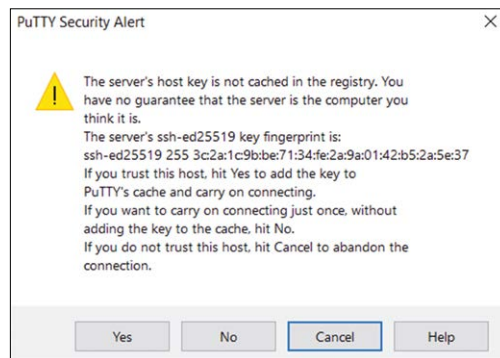


Figure 2.7 Click Yes to accept

You will be prompted to enter the username and password. Notice that the default username and password are:

username: **pi**
password: **raspberrypi**

You now have a terminal connection with the Raspberry Pi and you can type in commands, including sudo commands. You can use the cursor keys to scroll up and down through the commands you've previously entered in the same session. You can also run programs although not graphical programs.

2.4.1 Configuring Putty

By default, the Putty screen background is black with white foreground characters. In this book, we used a white background with black foreground characters, with the character size set to 12 points bold. The steps to configure Putty with these settings are given below. Notice that in this example the settings are saved with the name **RPI4** so that they can be recalled whenever Putty is re-started:

- Restart Putty
- Select **SSH** and enter the Raspberry Pi IP address
- Click **Colours** under **Window**
- Set the **Default Foreground** and **Default Bold Foreground** colours to black (Red:0, Green:0, Blue:0)
- Set the **Default Background** and **Default Bold Background** to white (Red:255, Green:255, Blue:255)
- Set the **Cursor Text** and **Cursor Colour** to black (Red:0, Green:0, Blue:0)
- Select **Appearance** under **Window** and click **Change** in **Font settings**. Set the font to **Bold 12**.
- Select **Session** and give a name to the session (e.g. RPI4) and click **Save**.
- Click **Open** to open the Putty session with the saved configuration
- Next time you re-start Putty, select the saved session and click **Load** followed by **Open** to start a session with the saved configuration

2.5 Remote access of the desktop

You can control your Raspberry Pi via Putty, and run programs on it from your Windows PC. This, however, will not work with graphical programs because Windows doesn't know how to represent the display. As a result of this, for example, we cannot run any graphical programs in the Desktop mode. We can get around this problem using some extra software. Two popular programs used for this purpose are VNC (Virtual Network Connection), and Xming. Here, we will be learning how to use VNC.

Installing and using VNC

VNC consists of two parts: VNC Server and VNC Viewer. VNC Server runs on the Raspberry Pi, and VNC Viewer runs on the PC. VNC server is already installed on your Raspberry Pi. You can start the server by entering the following command in the command mode:

```
pi$raspberrypi:~ $ vncserver :1
```

The steps to install and use VNC Viewer on your PC are given below:

- There are many VNC Viewers available, but the recommended one is TightVNC which can be downloaded from the following web site:

<https://www.tightvnc.com/download.php>

- Download and install **TightVNC** software for your PC. You will have to choose a password during the installation.
- Start **TightVNC Viewer** on your PC and enter the Raspberry Pi IP address (see Figure 2.8) followed by :1. Click **Connect** to connect to your Raspberry Pi.

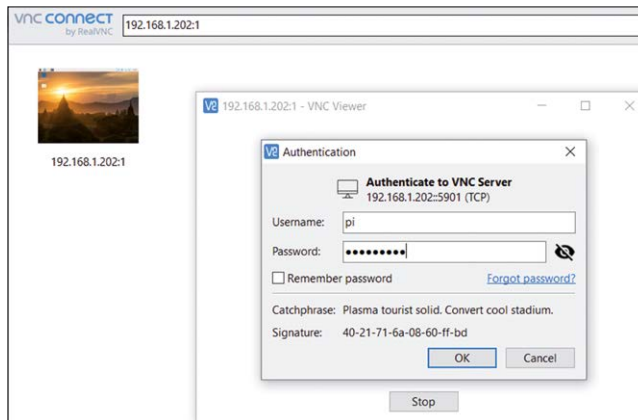


Figure 2.8 Start the TightVNC and enter the IP address

Figure 2.9 shows the Raspberry Pi Desktop displayed on the PC screen.

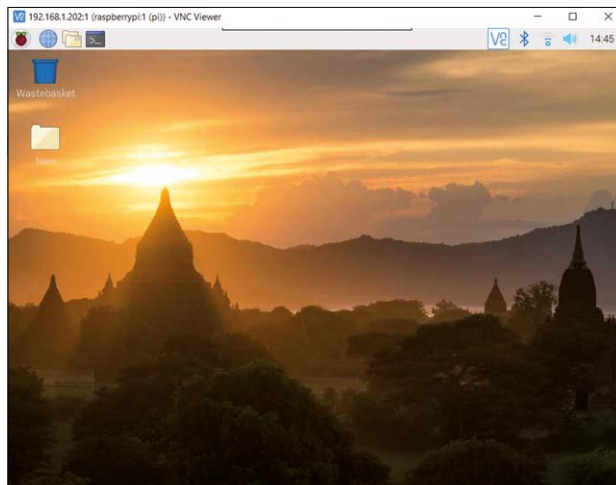


Figure 2.9 Raspberry Pi Desktop on the PC screen]

2.6 Summary

In this chapter, we have learned how to install the latest Raspberry Pi operating system on an SD card. Now that the software has been installed and our Raspberry Pi is working, in the next chapter, we will look at some of the important commands of the Raspberry Pi operating system.

Chapter 3 • Using the command line

3.1 Overview

Raspberry Pi operating system is based on a version of the Linux operating system. Linux is one of the most popular operating systems in use today. Linux is very similar to other operating systems, such as Windows and Unix. Linux is an open operating system based on Unix and has been developed collaboratively by many companies and universities since 1991. In general, Linux is harder to manage than some other operating systems like Windows but offers more flexibility and wider configuration options. There are several popular versions of the Linux operating system such as Debian, Ubuntu, Red Hat, Fedora and so on. In this chapter we shall be looking at some of the commonly used Raspberry Pi commands that we can enter from the command line. The commands entered by the user are shown in bold for clarity.

3.2 The Raspberry Pi directory structure

The Raspberry Pi directory structure consists of a single root directory, with directories and subdirectories under the root. Different operating system programs, applications, and user data are stored in different directories and subdirectories.

The root directory is identified by the "/" symbol. Under the root, we have directories named such as **bin**, **boot**, **dev**, **etc**, **home**, **lib**, **lost+found**, **media**, **mnt**, **opt**, **proc**, and many more. The important directory as far as the users are concerned is the **home** directory which contains subdirectories for each user of the system. The full path to the home directory is **/home/pi**. We can move to our home directory from any other directory by entering the command **cd ~**

Some useful directory commands are given below. Command **pwd** displays the user home directory:

```
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $
```

To show the directory structure, enter the command **ls /** as shown in Figure 3.1.

```
pi@raspberrypi:~ $ ls /
bin  dev  home  lost+found  mnt  proc  run  srv  tmp  var
boot  etc  lib  media      opt  root /sbin  sys  usr
```

Figure 3.1 List the directory structure

To show the subdirectories and files in our home directory, enter **ls** as shown in Figure 3.2.

```
pi@raspberrypi:~ $ ls
Desktop  Downloads  Music      Public      Videos
Documents  MagPi      Pictures  Templates
pi@raspberrypi:~ $
```

Figure 3.2 List of files in our home directory

The `ls` command can take a number of arguments. Some examples are given below:
To display the subdirectories and files in a single row:

```
pi@raspberrypi:~ $ ls -1
```

To display the file type, enter the following command. Note that directories have a "/" after their names, executable files have a "*" character after their names:

```
pi@raspberrypi:~ $ ls -F
```

To list the results separated by commas:

```
pi@raspberrypi:~ $ ls -m
```

We can mix the arguments as in the following example:

```
pi@raspberrypi:~ $ ls -m -F
```

Subdirectories are created using command **mkdir** followed by the name of the subdirectory. In the following example, subdirectory **myfiles** is created in our working directory (see Figure 3.3).

```
pi@raspberrypi:~ $ mkdir myfiles
pi@raspberrypi:~ $ ls
Desktop  Downloads  Music      Pictures  Templates
Documents  MagPi      myfiles    Public    Videos
pi@raspberrypi:~ $
```

Figure 3.3 Creating a subdirectory

Use command **rmdir** followed by the subdirectory name to remove a subdirectory.

3.3 File permissions

One of the important arguments used with the **ls** command is **"-l"** (lower case letter l) which displays the file permissions, file sizes, and when they were last modified. In the example in Figure 3.4, each line relates to one directory or file. Reading from right to left, the name of the directory or the file is on the right-hand side. The date the directory or the file was created is on the left-hand side of its name. Next comes the size in bytes. The characters at the beginning of each line are about the permissions. i.e. who is allowed to use or modify the file or the subdirectory.

```

pi@raspberrypi:~ $ ls -l
total 36
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Desktop
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Documents
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Downloads
drwxr-xr-x 2 pi pi 4096 Jun 20 17:55 MagPi
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Music
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Pictures
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Public
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Templates
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Videos
pi@raspberrypi:~ $

```

Figure 3.4 File permissions example

The permissions are divided into 3 categories:

- What the user (or owner, or creator) can do – called **USER**
- What the group owner (people in the same group) can do – called **GROUP**
- What everyone else can do – called **WORLD**

The first **pi** in the example shows who the user of the file (or subdirectory) is, and the second word **pi** shows the group name that owns the file (or subdirectory). In this example, both the user and the group names are **pi**.

The permissions can be analysed by breaking down the characters into four chunks for File type, User, Group, World. The first character for a file is "-", and for a directory, it is "d". Next comes the permissions for the User, Group, and World. The permissions are as follows:

- **Read permission (r)**: the permission to open and read a file or to list a directory
- **Write permission (w)**: the permission to modify a file, or to delete or create a file in a directory
- **Execute permission (x)**: the permission to execute the file (applies to executable files), or to enter a directory

The three letters **rwX** are used as a group and if there is no permission assigned then a "-" character is used.

As an example, considering the **Music** subdirectory, we have the following permission codes:

drwxr-xr-x which translates to:

```

d:      it is a directory
rwX:    user (owner) can read, write, and execute
r-x:    group can read and execute, but cannot write (e.g. create or delete)
r-x:    world (everyone else) can read and execute, but cannot write

```

The **chmod** command is used to change the file permissions. Before going into details of how to change the permissions, let us look and see what arguments are available in **chmod** for changing the file permissions.

The available arguments for changing file permissions are given below. We can use these arguments to add/remove permissions or to explicitly set permissions. It is important to realize that if we explicitly set permissions then any unspecified permissions in the command will be revoked:

u:	user (or owner)
g:	group
o:	other (world)
a:	all
+:	add
-:	remove
=:	set
r:	read
w:	write
x:	execute

To change the permissions of a file we type the **chmod** command, followed by one of the letters **u**, **g**, **o**, or **a** to select the people, followed by the **+** - or **=** to select the type of change, and finally followed by the filename. An example is given below. In this example, subdirectory **Music** has the user read and write permissions. We will be changing the permissions so that the user does not have read permission on this file:

```
pi@raspberrypi ~$ chmod -u -r Music
pi@raspberrypi ~$ ls -l
```

The result is shown in Figure 3.5.

```
pi@raspberrypi:~ $ chmod -u -r Music
pi@raspberrypi:~ $ ls -l
total 36
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Desktop
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Documents
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Downloads
drwxr-xr-x 2 pi pi 4096 Jun 20 17:55 MagPi
d----- 2 pi pi 4096 Jun 20 18:20 Music
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Pictures
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Public
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Templates
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Videos
pi@raspberrypi:~ $
```

Figure 3.5 File permissions of subdirectory Music

In the following example, **rwX** user permissions are given to subdirectory **Music**:

```
pi@raspberrypi ~$ chmod u+rwX Music
```

Figure 3.6 shows the new permissions of subdirectory **Music**.

```
pi@raspberrypi:~ $ chmod u+rwX Music
pi@raspberrypi:~ $ ls -l
total 36
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Desktop
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Documents
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Downloads
drwxr-xr-x 2 pi pi 4096 Jun 20 17:55 MagPi
drwx----- 2 pi pi 4096 Jun 20 18:20 Music
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Pictures
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Public
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Templates
drwxr-xr-x 2 pi pi 4096 Jun 20 18:20 Videos
pi@raspberrypi:~ $
```

Figure 3.6 New permissions of subdirectory Music

To change our working directory the command **cd** is used. In the following example we change our working directory to **Music**:

```
pi@raspberrypi ~$ cd /home/pi/Music
pi@raspberrypi ~/Music $
```

to go up one directory level, i.e. to our default working directory:

```
pi@raspberrypi ~/Music $ cd..

pi@raspberrypi ~$
```

to change our working directory to **Music**, we can also enter the command:

```
pi@raspberrypi ~$ cd ~/Music
pi@raspberrypi ~/myfiles $
```

to go back to the default working directory, we can enter:

```
pi@raspberrypi ~/Music $ cd ~
pi@raspberrypi ~$
```

to find out more information about a file we can use the **file** command. For example:

```
pi@raspberrypi ~$ file Music
Music: directory
pi@raspberrypi ~$
```

the **-R** argument of command **ls** lists all the files in all the subdirectories of the current working directory. An example is shown in Figure 3.7.

```
pi@raspberrypi:~ $ ls -R MagPi
MagPi:
MagPi82.pdf
pi@raspberrypi:~ $
```

Figure 3.7 Using command -R with ls

3.4 Help

Man

To display information on how to use a command, we can use the **man** command. As an example, to get help on using the **mkdir** command:

```
pi@raspberrypi ~$ man mkdir
MKDIR(1)

NAME
    Mkdir - make directories

SYNOPSIS
    Mkir [OPTION]...DIRECTORY...

DESCRIPTION
    Create the DIRECTORY(ies), if they do not already exist.

    Mandatory arguments to long options are mandatory for short options

    -m, --mode=MODE
        Set file mode (as in chmod), not a=rwx - umask
-----
-----
```

Enter **Cntrl+Z** to exit from the man display.

Help

The **man** command usually gives several pages of information on how to use a command. We can type **q** to exit the **man** command and return to the operating system prompt.

The **less** command can be used to display a long listing one page at a time. Using the up and down arrow keys, we can move between pages. An example is given below. Type **q** to exit:

```
pi@raspberrypi ~$ man ls | less
<display of help on using the ls command>
```

```
pi@raspberrypi ~$
```

3.5 Date, time, and calendar

To display the current date and time, the **date** command is used. Similarly, the **cal** command displays the current calendar. Figure 3.8 shows an example.

```
pi@raspberrypi:~ $ date
Wed 10 Jul 2019 11:53:55 AM BST
pi@raspberrypi:~ $ cal
      July 2019
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

pi@raspberrypi:~ $
```

Figure 3.8 Commands: **date** and **cal**

3.6 File operations

Copying a file

To make a copy of a file, use the command **cp**. In the following example, a copy of file **mytestfile.txt** is made and the new file is given the name **test.txt**:

```
pi@raspberrypi ~$ cp mytestfile.txt test.txt
```

Wildcards

We can use wildcard characters to select multiple files with similar characteristics. e.g. files having the same file-extension names. The "*" character is used to match any number of characters. Similarly, the "?" character is used to match any single character. In the example below all the files with extensions **".txt"** are listed:

```
pi@raspberrypi ~$ ls *.txt
```

The wildcard characters [a-z] can be used to match any single character in the specified character range. An example is given below which matches any files that start with letters **o**, **p**, **q**, **r**, **s**, and **t**, and with the **".txt"** extension:

```
pi@raspberrypi ~$ ls [o-t]*.txt
```

Renaming a file

You can rename a file using the **mv** command. In the example below, the name of file **test.txt** is changed to **test2.txt**:

```
pi@raspberrypi ~$ mv test.txt test2.txt
```

Deleting a file

The command **rm** can be used to remove (delete) a file. In the example below file **test2.txt** is deleted:

```
pi@raspberrypi ~$ rm test2.txt
```

The argument **-v** can be used to display a message when a file is removed. Also, the **-i** argument asks for confirmation before a file is removed. In general, the two arguments are used together as **-vi**. An example is given below:

```
pi@raspberrypi ~$ rm -vi test2.txt
rm: remove regular file 'test2.txt'? y
removed 'test2.txt'
pi@raspberrypi ~$
```

Removing a directory

A directory can be removed using the **rmdir** command:

```
pi@raspberrypi ~$ rmdir Music
```

Re-directing the output

The greater sign **>** can be used to re-direct the output of a command to a file. For example, we can re-direct the output of the **ls** command to a file called **ltest.txt**:

```
pi@raspberrypi ~$ ls > ltest.txt
```

The **cat** command can be used to display the contents of a file:

```
pi@raspberrypi ~$ cat mytestfile.txt
This is a file
This is line 2
pi@raspberrypi ~$
```

Using two greater signs **>>** adds to the end of a file.

Writing to the screen or to a file

The **echo** command can be used to write to the screen. It can be used to perform simple mathematical operations if the numbers and the operation are enclosed in two brackets, preceded by a **\$** character:

```
pi@raspberrypi ~$ echo $((5*6))
30
pi@raspberrypi ~$
```

The `echo` command can also be used to write a line of text to a file. An example is shown below:

```
pi@raspberrypi ~$ echo a line of text > lin.dat
pi@raspberrypi ~$ cat lin.dat
a line of text
pi@raspberrypi ~$
```

Matching a string

The **grep** command can be used to match a string in a file. An example is given below assuming that the file **lin.dat** contains sting **a line of text**. Notice that the matched word is shown in bold:

```
pi@raspberrypi ~$ grep line lin.dat
a line of text
pi@raspberrypi ~$
```

Head and tail commands

The `head` command can be used to display the first 10 lines of a file. The format of this command is as follows:

```
pi@raspberrypi ~$ head mytestfile.txt
.....
.....
pi@raspberrypi ~$
```

Similarly, the `tail` command is used to display the last 10 lines of a file. The format of this command is as follows:

```
pi@raspberrypi ~$ tail mytestfile.txt
.....
.....
pi@raspberrypi ~$
```

3.7 System and user information

These commands are useful as they tell us information about the system. Command **cat /proc/cpuinfo** displays information about the processor (command **cat** displays the contents of a file. In this example, the contents of the file **/proc/cpuinfo** is displayed). Since there are 4 cores in the Raspberry Pi 4, the display is in 4 sections. Figure 3.9 shows an example display, where only part of the display is shown here.


```
pi@raspberrypi:~ $ cat /proc/cpuinfo
processor       : 0
model name     : ARMv7 Processor rev 3 (v7l)
BogoMIPS      : 108.00
Features       : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt
vfpd32 lpae evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xd08
CPU revision   : 3

processor       : 1
model name     : ARMv7 Processor rev 3 (v7l)
BogoMIPS      : 108.00
Features       : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt
vfpd32 lpae evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xd08
CPU revision   : 3
```

Figure 3.9 Command: cat /proc/cpuinfo (only part of the output is shown)

Command **uname -s** displays the operating system Kernel name, which is Linux. Command **uname -a** displays complete detailed information about the Kernel and the operating system.

Command **cat /proc/meminfo** displays information about the memory on your Pi. Information such as the total memory and free memory at the time of issuing the command are displayed.

Command **whoami** displays the name of the current user. In this case, it displays **pi**.

A new user can be added to your Raspberry Pi using the command **useradd**. In the example in Figure 3.10, a user called Johnson is added. A password for the new user can be added using the **passwd** command followed by the username. In Figure 3.10, the password for user Johnson is set to **mypassword** (not displayed for security reasons). Notice that both the **useradd** and **passwd** are privileged commands and the keyword **sudo** must be entered before these commands. Notice that the **-m** option creates a home directory for the new user.

```
pi@raspberrypi:~ $ sudo useradd -m Johnson
pi@raspberrypi:~ $ sudo passwd Johnson
New password:
Retype new password:
passwd: password updated successfully
pi@raspberrypi:~ $
```

Figure 3.10 Commands: useradd and passwd

We can log in to the new user account by specifying the username and the password.

What software is installed on my Raspberry Pi

To find out what software is installed on your Raspberry Pi, enter the following command. You should get several pages of display:

```
pi@raspberrypi ~$ dpkg -l
.....
.....
pi@raspberrypi ~$
```

We can also find out if a certain software package is already installed on our computer. An example is given below which checks whether or not software called **xpdf** (PDF reader) is installed. In this example **xpdf** is installed and the details of this software are displayed:

```
pi@raspberrypi ~$ dpkg --s xpdf
Package: xpdf
Status: install ok installed
Priority: optional
Section: text
Installed-Size: 395
.....
.....
pi@raspberrypi ~$
```

If the software is not installed, we get a message similar to the following (assuming we are checking to see if a software package called **bbgd** is installed):

```
pi@raspberrypi ~$ dpkg -s bbgd
dpkg-query: package 'bbgd' is not installed and no information is available
.....
.....
pi@raspberrypi ~$
```

Super user commands

Some of the commands are privileged and only authorized persons can use them. Inserting the word **sudo** at the beginning of a command gives us the authority to use the command without having to log in as an authorized user.

3.8 Resource monitoring on Raspberry Pi

System monitoring is an important topic for managing usage of your Raspberry Pi. One of the most useful system monitoring commands is the **top**, which displays the current usage of system resources and displays which processes are running and how much memory and CPU time they are consuming.

Figure 3.11 shows a typical system resource display obtained by entering the following command (Enter **Ctrl+Z** to exit):

```
pi@raspberrypi ~$ top
pi@raspberrypi ~$
```

```

pi@raspberrypi: ~
top - 13:04:21 up 5:17, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 149 total, 1 running, 146 sleeping, 0 stopped, 2 zombie
%Cpu(s): 0.0 us, 5.2 sy, 0.0 ni, 94.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1939.5 total, 1353.7 free, 151.3 used, 434.5 buff/cache
MiB Swap: 100.0 total, 100.0 free, 0.0 used, 1669.8 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 3419 pi          20   0 10320 2876 2512  R   11.8   0.1   0:00.06 top
    1 root        20   0 33724 7904 6288  S    0.0   0.4   0:03.78 systemd
    2 root        20   0    0    0    0  S    0.0   0.0   0:00.01 kthreadd
    3 root         0 -20    0    0    0  I    0.0   0.0   0:00.00 rcu_gp
    4 root         0 -20    0    0    0  I    0.0   0.0   0:00.00 rcu_par_gp
    8 root         0 -20    0    0    0  I    0.0   0.0   0:00.00 mm_percpu+
    9 root        20   0    0    0    0  S    0.0   0.0   0:00.06 ksoftirqd+
   10 root        20   0    0    0    0  I    0.0   0.0   0:00.93 rcu_sched
   11 root        20   0    0    0    0  I    0.0   0.0   0:00.00 rcu_bh
   12 root        rt    0    0    0    0  S    0.0   0.0   0:00.02 migration+
   13 root        20   0    0    0    0  S    0.0   0.0   0:00.00 cpuhp/0
   14 root        20   0    0    0    0  S    0.0   0.0   0:00.00 cpuhp/1
   15 root        rt    0    0    0    0  S    0.0   0.0   0:00.01 migration+
   16 root        20   0    0    0    0  S    0.0   0.0   0:00.04 ksoftirqd+
   19 root        20   0    0    0    0  S    0.0   0.0   0:00.00 cpuhp/2
   20 root        rt    0    0    0    0  S    0.0   0.0   0:00.01 migration+
   21 root        20   0    0    0    0  S    0.0   0.0   0:00.10 ksoftirqd+

```

Figure 3.11 Typical system resource display

Some of the important points in Figure 3.11 are summarized below (for lines 1 to 5 of the display):

- There are a total of 149 processes in the system
- Currently, only one process is running, 146 processes are sleeping, and 2 processes are in Zombie state
- The percentage CPU utilization is 0.0us for user applications (us)
- The percentage CPU utilization for system applications is 5.2 (sy)
- There are no processes requiring more or less priority (ni)
- 94.8% of the time the CPU is idle (id)
- There are no processes waiting for I/O completion (wa)
- There are no processes waiting for hardware interrupts (hi)
- There are no processes waiting for software interrupts (si)
- There is no time reserved for a hypervisor (st)
- The total usable memory is 1939.5MB, of which 151.3MB bytes are in use, 1353.7MB are free, and 434.5MB are used by buffers/cache
- Line 5 displays the swap space usage

The process table gives the following information for all the processes loaded to the system:

- PID: the process ID number
- USER: the owner of the process
- PR: priority of the process
- NI: the nice value of the process
- VIRT: the amount of virtual memory used by the process
- RES: the size of the resident memory
- SHR: shared memory the process is using
- S: process status (sleeping, running, zombie)
- %CPU: the percentage of CPU consumed
- %MEM: percentage of RAM used

- **TIME+:** total CPU time the task used
- **COMMAND:** The actual name of the command

The **ps** command can be used to list all the processes used by the current user. An example is shown in Figure 3.12.

```
pi@raspberrypi:~ $ ps
  PID TTY          TIME CMD
 1290 pts/0        00:00:00 bash
 2070 pts/0        00:00:01 Xtightvnc
 2074 pts/0        00:00:00 xstartup
 2077 pts/0        00:00:00 lxsession
 2120 pts/0        00:00:00 openbox
 2124 pts/0        00:00:08 lxpanel
 2126 pts/0        00:00:00 pcmanfm
 2171 pts/0        00:00:00 sh <defunct>
 3419 pts/0        00:00:03 top
 3457 pts/0        00:00:00 ps
pi@raspberrypi:~ $
```

Figure 3.12 Command: **ps**

the command **ps -ef** gives a lot more information about the processes running in the system.

Killing a process

There are many options for killing (or stopping) a process. A process can be killed by specifying its PID and using the following command:

```
pi@raspberrypi ~$ kill -9 <PID>
```

Disk usage

The disk free command **df** can be used to display the disk usage statistics. An example is shown in Figure 3.13. option **-h** displays in human-readable form.

```
pi@raspberrypi:~ $ df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/root        28683772 4397836  22805824  17% /
devtmpfs         860920      0    860920   0% /dev
tmpfs            993016      0    993016   0% /dev/shm
tmpfs            993016     8684   984332   1% /run
tmpfs             5120        4     5116   1% /run/lock
tmpfs            993016      0    993016   0% /sys/fs/cgroup
/dev/mmcblk0p6  258094     39984   218111  16% /boot
tmpfs            198600      0   198600   0% /run/user/1000
```

Figure 3.13 Command: **df**

3.9 Shutting down

Although you can disconnect the power supply from your Raspberry Pi when you finish working with it, it is not recommended since there are many processes running on the system and it is possible to corrupt the file system. It is much better to shut down the system in an orderly manner.

The following command will stop all the processes and make the file system safe and then turn off the system safely:

```
pi@raspberrypi ~$ sudo halt
```

The following command stops and then re-starts the system:

```
pi@raspberrypi ~$ sudo reboot
```

The system can also be shut down and then restarted after a time by entering the following command. Optionally, a shutdown message can be displayed if desired:

```
pi@raspberrypi ~$ shutdown -r <time> <message>
```

3.10 Summary

This Chapter has described the use of a number of important Linux commands. You should be able to get further information on each command and other Linux commands from the internet and from other books on Raspberry Pi and Linux.

In the next chapter, we shall see how to install Node-RED software on our Raspberry Pi computer.

Chapter 4 • Installing Node-RED software on Raspberry Pi

4.1 Overview

In the last chapter, we have seen how to use some of the common Raspberry Pi command line commands from a terminal. In this chapter, we will be installing Node-RED software on our Raspberry Pi SD card.

4.2 Raspberry Pi Node-RED installation

Node-RED is already installed on the Raspbian Buster operating system of Raspberry Pi (you will find that it may also be installed if you are using an earlier version of the operating system, e.g. the Jessie-Version 2017-04-010, version 0.15.3 or a newer version, and it can be started immediately).

If Node-RED is not installed on your Raspberry Pi, you can easily install it by entering the following command:

```
pi@raspberrypi:~ $ sudo apt-get install nodered
```

At the time of writing this book, the default version of Node-RED pre-installed with the Raspbian Buster operating system was v0.20.6 and this was not the latest version. If Node-RED is already installed on your Raspberry Pi, you can upgrade to the latest version (recommended) by entering the following command (see link: <https://nodered.org/docs/getting-started/raspberrypi>):

```
pi@raspberrypi:~ $ bash <(curl -sL https://raw.githubusercontent.com/  
node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

After upgrading Node-RED, the latest version at the time of writing this book was v1.0.3 and this is the version used in all examples in this book.

You can start Node-RED by entering the following command in the command mode (characters entered by the user are in bold for clarity):

```
pi@raspberrypi:~ $ node-red-start
```

When started, you should see some messages scrolling in your screen, where part of the messages is shown in Figure 4.1.

```
pi@raspberrypi:~ $ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.202:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use  node-red-stop                to stop Node-RED
Use  node-red-start              to start Node-RED again
Use  node-red-log                to view the recent log output
Use  sudo systemctl enable nodered.service to autostart Node-RED at every boot
Use  sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
```

Figure 4.1 Part of the messages displayed when Node-RED started

A list of the important commands are given below:

```
node-red-start:      start Node-RED
node-red-stop:     stop Node-RED
node-red-log:      view the recent log output
sudo systemctl enable nodered.service: autostart Node-RED after every
                                           boot
sudo systemctl disable nodered.service: disable autostart after every boot
```

Notice that this is a service type application and therefore it can be accessed from anywhere on the network and from other computers as well. To do this, we have to know the IP address of our Raspberry Pi. This can be found by entering the following command at the command prompt:

```
pi@raspberrypi:~ $ ifconfig
```

The IP address of our Raspberry Pi in this example is 192.168.1.202 as shown in Figure 4.2 (only part of the output is shown here).

```
--
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.202 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fd83:29d0:5e0f:1:2fe0:b5bf:1692:5d3a prefixlen 64 scopeid 0x0<gl
obal>
    inet6 fe80::7abd:1a67:fe80:e3ba prefixlen 64 scopeid 0x20<link>
    ether dc:a6:32:00:e4:29 txqueuelen 1000 (Ethernet)
    RX packets 1135 bytes 114157 (111.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2207 bytes 2964048 (2.8 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 4.2 Finding the IP address of our Raspberry Pi

We can now enter the following command on our web browser to display the Node-RED startup screen:

<http://192.168.1.202:1880>

When started, you should see the Node-RED startup screen as in Figure 4.3. When Node-RED is started as a service, pressing Cntrl+C does not stop the service as it keeps running in the background.

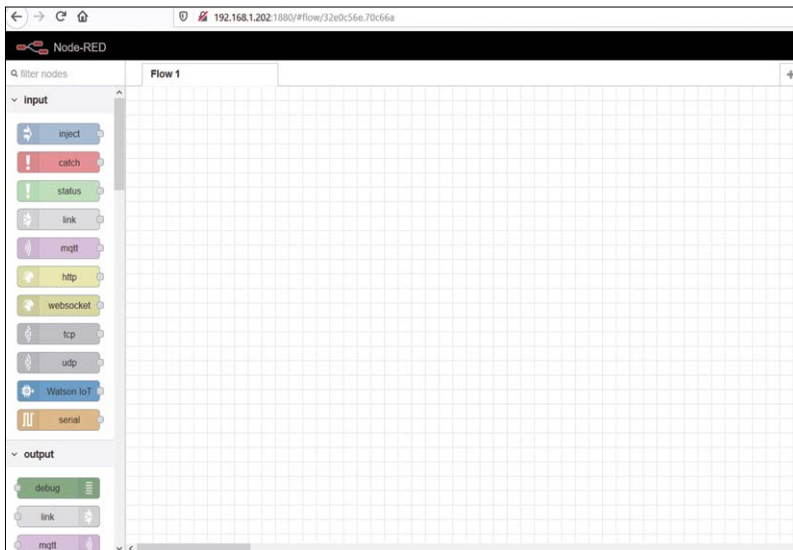


Figure 4.3 Node-RED startup screen

Notice that, if you are using a browser on the same computer that Node-RED is running, then you can access Node-RED using the following URL:

<http://localhost:1880>

4.3 Node-RED interface to external world

Node-RED is normally accessed from a web browser. Figure 4.4 shows the Node-RED interface to the external world. Raspberry Pi accesses the Node-RED software through the local Wi-Fi router (or Internet). Sensors, actuators, and any other hardware are interfaced directly to the Raspberry Pi GPIO ports through the 40-pin header mounted on Raspberry Pi. Raspberry Pi communicates with the external hardware by receiving commands from Node-RED through the local Wi-Fi router.

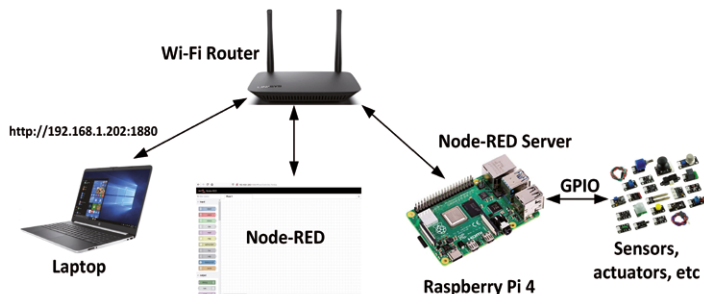


Figure 4.4 Node-RED Interface to External World

The screenshot shows the Node-RED web interface. On the left, a sidebar contains two panels: 'Categories' and 'Nodes'. The 'Nodes' panel lists various node types such as 'input', 'link', 'status', 'link', 'output', 'http', 'websocket', 'tcp', 'udp', 'mqtt', 'redis', and 'serial'. The main workspace, labeled 'Flow 1', is a large grid area. On the right, a 'Menu' panel displays information about the selected flow, including its name 'Flow 1', status 'Enabled', and a description. Labels with arrows point to these specific components: 'Categories', 'Nodes', 'Flow Area (Workspace)', 'Menu', and 'Description'.

The node categories, also called the Node Palettes consists of following sections:

- ## 4.5 Project 1 – Hello World!

- Start Node-RED on your Raspberry Pi:

- Enter the following URL into your web browser to display the Node-RED screen (you will have to enter the IP address of your own Raspberry Pi):

- From the node palette, Click on **inject** node on the left-hand side and drag it to the workspace.

- 44

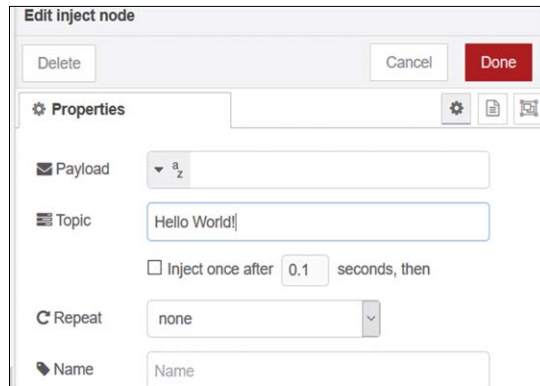


Figure 4.6 Enter Hello World!

- Click **Done**
- Add node debug by clicking on **debug** node and dragging it to the right-hand side of the inject node already in the workspace
- Wire the two nodes together. Place the mouse cursor over the **inject** nodes output port (a small gray square on the right-hand side of the node), then left-click and drag a wire over to the input port of the Debug node. A wire should now connect the two nodes as shown in Figure 4.7



Figure 4.7 Connect the two nodes together

- Click the **Deploy** button at the top right-hand side of the screen
- Click the small gray square on the right-hand side of node **debug** to activate the node. You should see the message **Successfully activated: debug** displayed by the console window
- Select the debug tab on the top right-hand side.
- Click the **inject** nodes button, which is the small square coming out from the left-hand side of the node. Clicking the button will inject the message into the flow and to the debug node. The output should be displayed on the debug window as shown in Figure 4.8

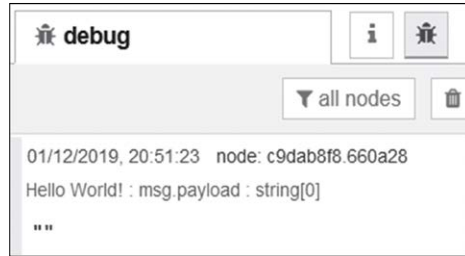


Figure 4.8 Display the output at debug window

Notice that, if you place a wrong node on the workspace, you can click on the node and then use the keyboard delete button to remove it.

In this example, we first created an inject node. The inject node can be used to manually trigger a flow by clicking the nodes button within the workspace. It can also be used to automatically trigger flows at regular intervals. We then stored the string **Hello World!** in this node. Next, we created a debug node and connected the two nodes together and activated the debug node. By triggering the inject node, the string is sent to the debug node and is displayed in the debug window. The Debug node can be used to display messages in the Debug sidebar within the editor. In addition to the displayed messages, the debug sidebar includes information about the time the message was received and also the ID of the Debug node that sent the message. Clicking on the ID reveals the node that sent the message.

Modified program

We can modify the flow program, for example, to display the string every 5 seconds. The steps to do this are given below:

- Double click on **inject** node
- Set the Repeat Interval to every 5 seconds (Figure 4.9) and click **Done**

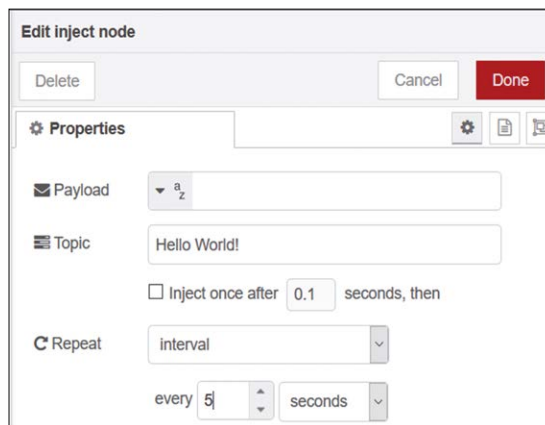


Figure 4.9 Set the repeat interval to 5 seconds

- Click **Deploy**
- Activate the **debug** node.
- Click the **inject** nodes button. You should see the message displayed every 5 seconds as in Figure 4.10

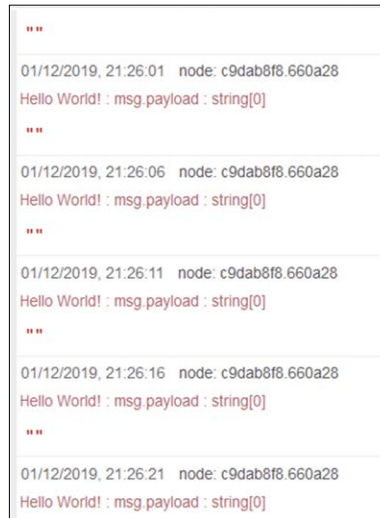


Figure 4.10 Displaying the message every 5 seconds

4.6 Project 2 – Date and time

In this program, we will be using the **function** node to return the current date and time to the debug window. The steps are as follows:

- From the node palette, Click on **inject** node on the left-hand side and drag it to the workspace.
- Double click on this node, click on **Payload**, select **String** and enter the following string **THE DATE AND TIME ARE:.** Select the Repeat interval to 1 second as shown in Figure 4.11.

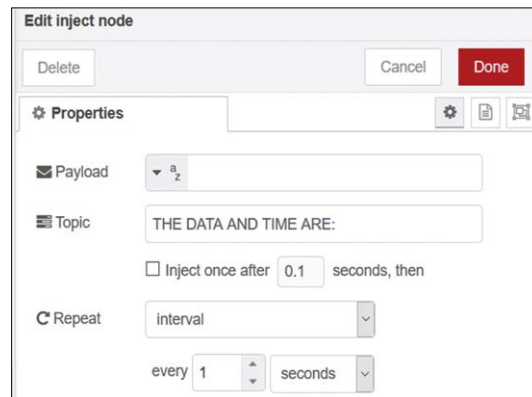


Figure 4.11 The inject node

- Click **Done**
- Add node function by clicking on **function** node and dragging it to the right-hand side of the inject node already in the workspace
- Wire the two nodes together. Click on the function node and enter the following as shown in Figure 4.12. Click **Done**:

```
msg.payload=new Date();
return msg;
```

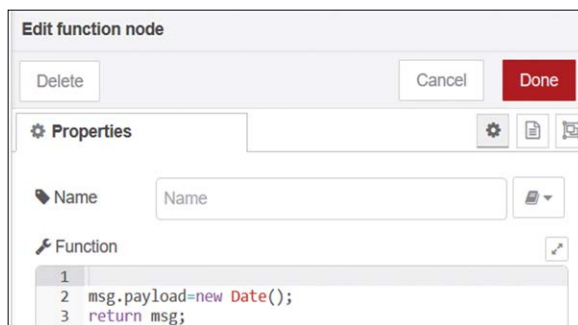


Figure 4.12 Function to get the data and time

- Add node debug by clicking on **debug** node and dragging it to the right-hand side of the inject node already in the workspace.
- Join the three nodes together as shown in Figure 4.13.



Figure 4.13 Join the three nodes

- Click the small gray square at the right-hand side of node **debug** to activate the node. You should see the message **Successfully activated: debug** displayed by the console window
- Click the **inject** nodes button. You should see the date and time displayed every second in the following format (see Figure 4.14):

```
DATE AND TIME ARE:
"Mon Dec 02 2019 15:11:03 GMT+0000
(Greenwich Mean Time)"
```

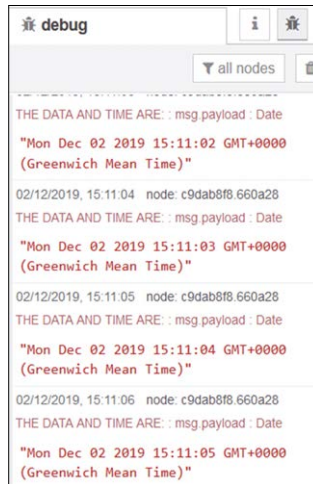


Figure 4.14 Displaying the message every 5 seconds

We have learned in this example that the function node allows JavaScript code to be run against the messages that are passed through it. A **function** node receives data from its left, processes the data and outputs from its right.

A function node can have multiple outputs. To create more than one output, double click on the function node and set **Outputs** at the bottom left corner to the required value. Additionally, a function node can create a library by saving your functions. This is done by double-clicking the function node and then clicking the **book** icon next to **Name**. Here, you will be given options to **Open** an existing library or to **Save** the function in a library.

Flow names

We can add new flows by clicking the **+** sign at the top right-hand side of the screen. The name of a flow can be changed as follows:

- Click the menu on the far top right-hand side of the screen
- Select **Flows -> Rename** as shown in Figure 4.15

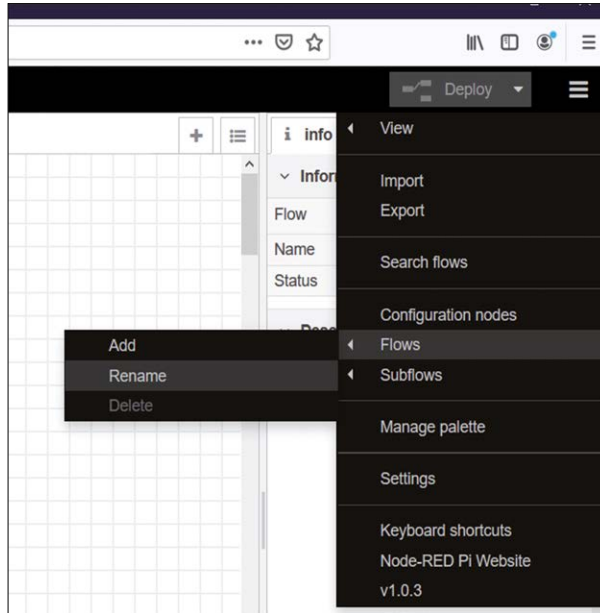


Figure 4.15 Renaming a flow

- Change the name of the existing flow as required and then click **DONE**.

Modified program

We can modify the function in the above flow program, for example just to display the local time in the following format:

```
Time = hh:mm:ss
```

The steps are given below:

- Insert an **inject** node to repeat every second with the string THE CURRENT TIME
- Insert a **function** node and enter the following statements inside this function node and click **DONE**:

```
var LocalTime = new Date();
var Tim = LocalTime.toLocaleTimeString();
msg.payload = msg.payload + "Time = " + Tim;
return msg;
```

- Insert a **debug** node.
- Join all three nodes. Click Deploy and activate the **debug** node

- Click the **inject** nodes button. You should see the time updating every second in the Debug window as in the format shown in Figure 4.16.

```
02/12/2019, 18:55:18 node: c9dab8f8.660a28
THE CURRENT TIME : msg.payload : string[15]
"Time = 18:55:17"
```

Figure 4.16 Displaying the local time in Debug window

It is important to realize that the programming in Node-RED is based on JavaScript which is based on objects. Variables are represented by the keyword **var**. For example, a person may be represented as follows:

```
var person = {firstname: "Dogan", surname: "Ibrahim", age: 50, height: 160};
```

the elements of person can, for example, be accessed as follows:

```
person.firstname or person.surname, etc.
```

The Node-RED message object is called **msg** and it has the following properties: payload, topic, and **_msgid**.

payload: this property by default stores the payload of the message. The payload can take the following values (see link: <https://nodered.org/docs/user-guide/messages>):

- Boolean - true, false
- Number - eg 0, 123.4
- String - "hello world"
- Array - [1,2,3,4,5]
- Object - { "a": 1, "b": 2 }
- Null

By default, the Debug node displays the **msg.payload** property of a message but can be configured to display any property or the whole message.

When using a function node, we can store a new text string in variable msg as follows:

```
var msg = {payload: "This is a string"};
```

or, we can use the following statement:

```
msg.payload = "This is a string";
```

topic stores the parent topic of a message

_msgid: this is an ID that can be used to identify and track the progress of the message through a flow.

4.7 Project 3 – Temperature conversion

In this program, we will be introducing the **change** node. This is a temperature conversion program. The program converts degrees Celsius to degrees Fahrenheit. As an example, 10°C is converted into °F and is displayed in the Debug window.

The steps are as follows:

- Create an **inject** node with the string **Temperature Conversion**
- Create a **change** node and set the msg.payload to 10 (this is the degrees Celsius) as shown in Figure 4.17. Click **DONE**

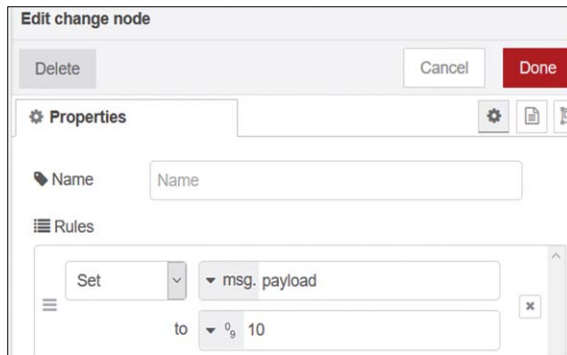


Figure 4.17 Create a change node

- Create a **function** node with the following statements to convert °C to °F:

```
var F = msg.payload;
var C = F*1.8+32;
msg.payload = F + "C = " + C.toString() + "F";
return msg;
```

- Create a **debug** node and join all four nodes together as shown in Figure 4.18.



Figure 4.18 Join the nodes together

- Click **Deploy**
- Activate the **debug** node and then click the **inject** nodes button. You should see 10°C is converted into 50°F in the Debug node as shown in Figure 4.19

```
02/12/2019, 21:14:10 node: c9dab8f8.660a28
Temperature Conversion : msg.payload : string[9]
"10C = 50F"
```

Figure 4.19 Conversion displayed by the Debug window

In this example, we have used a **change** node. The change node provides the following functionalities:

- Set a property to a value
- Change a string property (by performing search and replace)
- Delete a property
- Move a property

With the set operation, we can set a value to `msg.payload`. This value can be a string or a number as in the above example, or it can come from another message.

Saving function to a library

Figure 4.20 shows how the statements in the function to convert °C to °F can be saved in a library called **TempConv**. To do this, click the book icon in the function node and then click **Save to Library**, and click **Save**

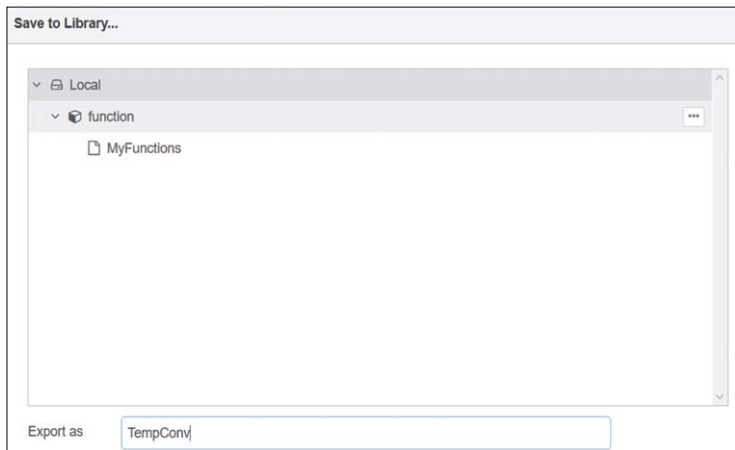


Figure 4.20 Saving function to a library

Loading function from a library

Figure 4.21 shows how a library item can be loaded from the library. In this example, library item **TempConv** which was saved earlier is loaded into the function. Open the library by clicking the **book** icon in **function** node, select **TempConv** and click **Load**.

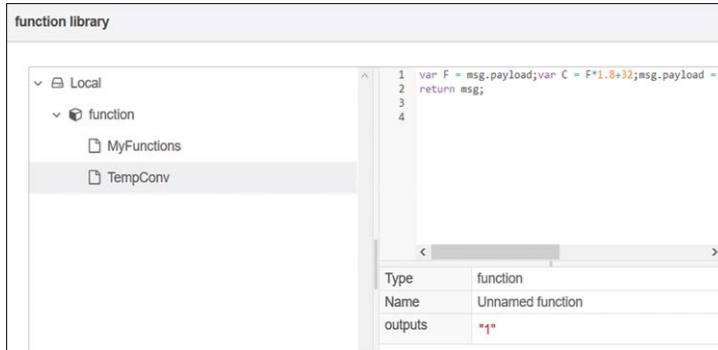


Figure 4.21 Loading a library item

There are many nodes available in the standard Node-RED, and it is not possible to look at all of them in detail in this book. There are also third-party nodes, created and distributed by several firms. We will only look at the nodes and make ourselves familiar with the nodes that we will be using in projects with our Raspberry Pi. We can summarize that the nodes are the fundamental parts of node-RED and some of their application areas are:

- Accessing the GPIO pins of the Raspberry Pi. This enables us to undertake projects using various sensors and actuators connected to the Raspberry Pi
- Sending and receiving data packets from the Internet. This enables us, for example, to carry out remote control operations, for example, to control equipment over the Wi-Fi
- Machine to machine communication
- Sensing and receiving data from the Cloud. This application enables us to develop IoT based applications with the help of Node-RED and Raspberry Pi
- Learning and enhancing our programming skills

4.8 Importing and exporting flow programs

We can import and export flow programs between computers using the export and import features of Node-RED. Flows are exported as a JSON file, and also imported as a JSON file.

4.8.1 Exporting flows

Press **CTRL** and at the same time Click each node in the flow that you want to export. You will see the clicked nodes as highlighted. If you wish to export all nodes in the workspace, then click **CTRL+A**. Then, click **Menu -> Export**. You will be presented with a window as in Figure 4.22. Click **Copy to clipboard**. There are three tabs on the window with the **selected nodes** tab selected. You can if you wish, change it to export the entire flow or all flows in the workspace, by selecting **current flow** or the **all flows** tabs respectively. You should now open a file in a text editor and paste the contents of the JASON flow data into this file using **Paste**, or **CTRL+V**. Finally, give a name to the file and save it in a folder of your choice.

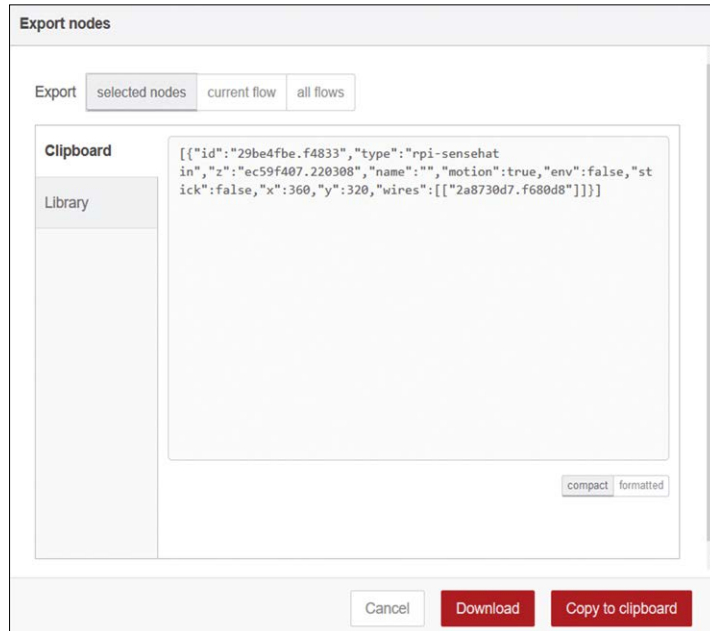


Figure 4.22 Exporting a flow

4.8.2 Importing aflow

A JASIN formatted flow file must be opened in a text editor and the flow data copied into it using **CTRL+C**. Then you should go to **Menu->Import** and paste the file from the clipboard into the window. The new flow can be imported into an existing flow, or into a new flow. Finally, click the **Import** button to import the flow. If you find that the Import button doesn't work (e.g. the button colour is not changed), then it is probable that the file you are trying to import is not a valid JSON file.

4.9 Copying nodes in the same workspace

There may be applications where you may want to copy a node in Node-RED. To do this, you simply press **CTRL** and click the nodes to be copied, or press **CTRL+A** to select all the nodes in the workspace. Then, click **CTRL+C** to copy the nodes. Move the cursor where you want the nodes to be copied to and press **CTRL+V** to paste the nodes into the new location in the workspace.

4.10 Core nodes

When Node-RED is installed on Raspberry Pi, it starts with categories of nodes known as the core nodes (see Figure 4.5). We will now briefly look at the nodes in each category.

4.10.1 Input nodes

inject: this node injects a timestamp or text into a message. The node can be configured either to inject manually or at regular intervals.

catch: If a node gives an error while handling a message, it is probable that the flow will halt. The catch node can be used to catch the errors and return messages about the error.

mqtt in: this node connects to an MQTT broker and subscribes to messages from the specified topic

http in: this node creates an HTTP end-point for creating web services

websocket: this node provides an endpoint for a browser so that a Websocket connection can be established with Node-RED

tcp: this node is used to establish TCP communications on a specified port. It accepts incoming TCP requests

udp: this node is used to establish communication using the UDP protocol. It accepts incoming UDP packets

serial in: this node reads serial data from the serial port of the local device. The node can be configured to read at specific times.

4.10.2 Output nodes

debug: This node is used to view messages on the Debug window. The node can be configured to display either the entire message or just the msg.payload

mqtt out: this node connects to an MQTT broker and publishes messages

http response: this node sends responses back to the HTTP requests received from HTTP input node.

http request: this node sends HTTP requests and returns the response

websocket: this node sends msg.payload to the configured websocket

tcp: this node replies to a configured TCP port.

udp: this node sends UDP messages to the configured UDP host where the IP address and port number are specified. Broadcast is supported

serial out: this node sends data to the defined serial port of the local device. The node can be configured to send some control characters such as newline after a message

4.10.3 Function nodes

function: this is a processing node, used to write JavaScript compatible functions

template: this node is useful for constructing messages and configuration files where name:value pairs are inserted into the template

delay: this node delays messages by a random or specific time

trigger: this node can be used as a watchdog timer. It can also be used to create two output messages with time intervals when an input message is received

comment: this node is used to insert comments

http request: this node is used to construct and then send an HTTP request to a given URL

tcp request: this node sends msg.payload to a TCP server and expects a response. The node is configurable, for example, it can be configured to wait for a specific character, or to return immediately

switch: this node is used to route messages based on their properties

change: this node can be used to set, change, or delete the properties of incoming messages

range: this node maps numerical input data to new output data

csv: this node parses msg.payload and converts to or from CSV format. The input can be a string in which case a JavaScript object is outputted. If on the other hand, the input is a JavaScript object, then a CSV formatted string is outputted

html: this node is used to extract data from an html type document in msg.payload

json: this node converts to or from a JSON object and JavaScript object

xml: this node converts to or from XML format and JavaScript object

random: this node generates a random number between a high and a low value

smooth: this node is used for various functions such as max, min, mean, high, and low pass filter

rbe: this is the rbe (Report By Exception) node which generates message if its input is different from the previous input

4.10.4 Social nodes

email in: this node is used to read new emails as they arrive at the localhost. It can be configured to read repeatedly

twitter in: this node is used to return tweeter messages

email out: this node is used to send email messages

twitter out: this node Tweets the msg.payload on the configured user account. Text and binary (image) messages can be sent

4.10.5 Storage nodes

tail: this node tails a file and injects the contents into the flow

file in: this node reads the specified file and sends its contents as msg.payload

file: this node writes the msg.payload to the specified file

4.10.6 Analysis nodes

sentiment: this node analyses the msg.payload and scores incoming words using the AFINN-165 wordlist, and attaches a sentiment.score property to the msg

4.10.7 Advanced nodes

watch: this node watches a folder for changes and sends events when files are added, changed, created or deleted

feedparse: this node monitors an RSS/Atom feed for new entries and if there are new entries, it delivers them as messages

exec: this node calls a system command and provides stdout, stderr, and the return code

4.10.8 Raspberry Pi nodes

rpi gpio in: this is the Raspberry Pi input node. Depending on the state of the input, this node generates a msg.payload with either a 0 or 1. We shall be using this node in the Raspberry Pi based projects in later Chapters

rpi gpio out: this is the Raspberry Pi output node. The selected hardware pin of Raspberry Pi is set HIGH or LOW depending on whether msg.payload is 0 or 1. We shall be using this node in the Raspberry Pi based projects in later Chapters

rpi mouse: this is the Raspberry Pi mouse button node. The node generates a 0 or 1 when the selected mouse button is clicked and released. A USB mouse is required

rpi keyboard: this node is used to capture keystrokes from a USB keyboard

Sense HAT: this is the Raspberry Pi Sense HAT input (or output) node. There are two nodes with the same name: one for the input, and the other one for the output

4.11 Project 4 – Dice number

In this program, we will be introducing the **random** node. The program will display a random number between 1 and 6 every time it is clicked to start. The steps are as follows:

- Create an **injection** node with the title **Your Chance**
- Create a **random** node and name it as Dice. Double click to edit it and set to generate whole integer numbers between 1 and 6 as shown in Figure 4.23. Click **Done**

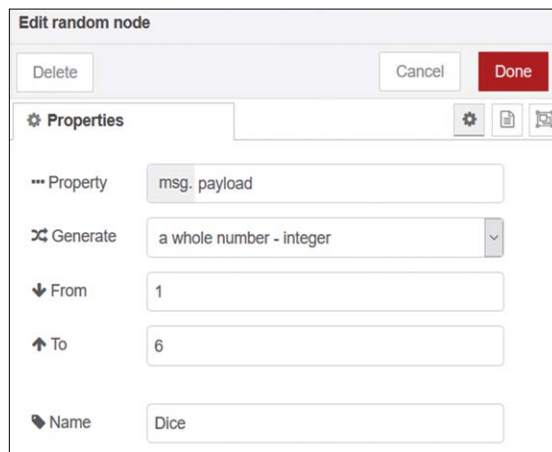


Figure 4.23 Generate random numbers between 1 and 6

- Create a **debug** node
- Join all three nodes as shown in Figure 4.24 and click **Deploy**



Figure 4.24 Join all three nodes

- Click the **inject** nodes button. You should see a number between 1 and 6 displayed in the Debug window. Every time you click the inject nodes button, a random number will be generated and displayed between 1 and 6 as shown in Figure 4.25

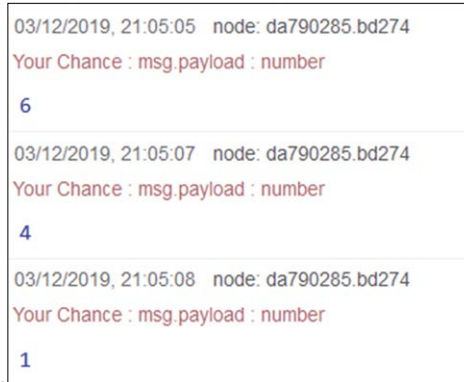


Figure 4.25 The Debug window

4.12 Project 5 – Double dice numbers

In many dice games such as backgammon, two dice are thrown at any time by each player before making a move. The program given in the previous project is modified in this project so that two random dice numbers are generated and displayed every time the **inject** nodes button is clicked. This program uses two **random** nodes and a **join** node and the steps to design the flow are given below:

- Create an **injection** node with the title **Your Chance**
- Create two **random** nodes with the names **Dice1** and **Dice2**. Double click to edit them and set to generate whole integer numbers between 1 and 6 for both nodes as in the previous project.
- Create a **join** node with the name **join payloads**. Double click this node and edit as shown in Figure 4.26. Notice that the message parts are set to 2 and are separated with a space character. Click **Done**

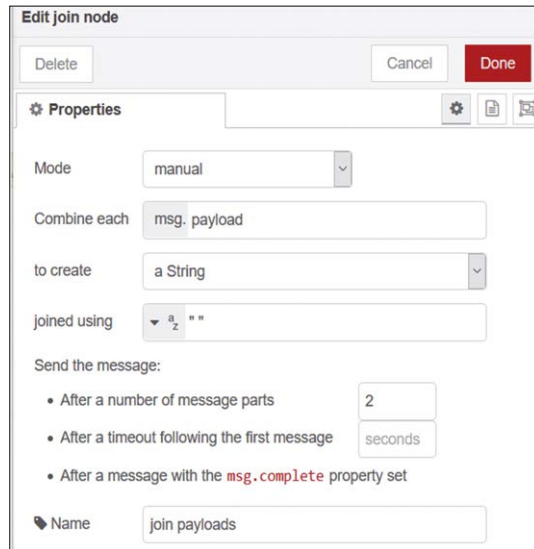


Figure 4.26 Edit the join node

- Create a debug node as before and activate it.
- Join the five nodes as shown in Figure 4.27, and click **Deploy**

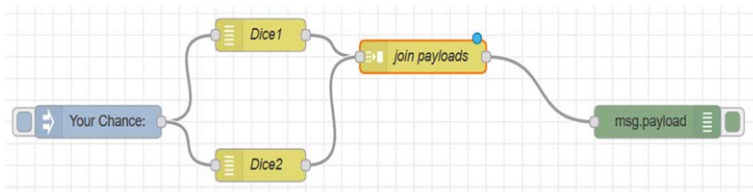


Figure 4.27 Join the five nodes together

- Click the **inject** nodes button to generate two random numbers next to each other in the Debug window as shown in Figure 4.28.

```
03/12/2019, 21:59:27 node: 3d66beb2.0791ca
Your Chance : msg.payload : string[5]
"4" "6"

03/12/2019, 21:59:28 node: 3d66beb2.0791ca
Your Chance : msg.payload : string[5]
"5" "4"

03/12/2019, 21:59:28 node: 3d66beb2.0791ca
Your Chance : msg.payload : string[5]
"2" "2"
```

Figure 4.28 The Debug window

Notice here that, if the **After a number of message parts** in node join was set to 1, then the two dice numbers would not have been displayed next to each other as shown in Figure 4.29.

```
03/12/2019, 22:20:59 node: 3d66beb2.0791ca
Your Chance : msg.payload : string[1]
"3"

03/12/2019, 22:20:59 node: 3d66beb2.0791ca
Your Chance : msg.payload : string[1]
"1"
```

Figure 4.29 Displaying the two numbers separately

4.13 Project 6 – Unit conversions - Multiple inputs to a function

In this flow program, we will have 3 inputs to a function. The flow has 3 inject nodes named `inchToCm`, `cmToInch` and `mToCm` where,

inchToCm: this node injects a number to the function to convert inches to cm. In this example, the value (topic) is set to 12 inches for demonstration purposes

cmToInch: this node injects a number to the function to convert cm to inches. In this example, the value (topic) is set to 10 cm for demonstration purposes

mToCm: this node injects a number to the function to convert meters to cm. In this example, the value (topic) is set to 2 m for demonstration purposes

The steps are as follows:

- Create 3 **inject** nodes with the following characteristics:

Node	Payload	Topic
1	inchToCm	12
2	cmToinch	10
3	mToCm	2

- Create a **function** node named **Conv**, and enter the following code inside the function (see Figure 4.30) and click **Done**:

```
var conv = msg.payload
var number = msg.topic

if(conv == "inchToCm")
{
    msg.payload = number+"inch = "+number*2.54+"cm";
}
else if(conv == "cmToinch")
{
    msg.payload = number+"cm = "+ number/2.54+"inch";
}
else if(conv == "mToCm")
{
    msg.payload = number+"m = "+number *100+"cm";
}
return msg;
```

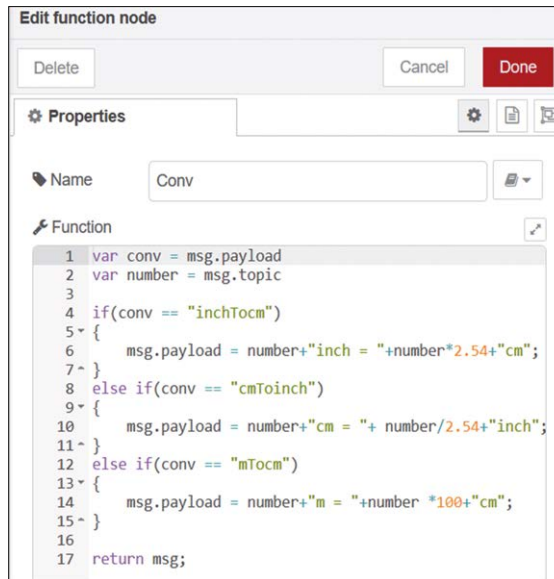


Figure 4.30 Create a function node

- Create a **debug** node and then connect all 5 nodes together as shown in Figure 4.31. Then, click **Deploy**.

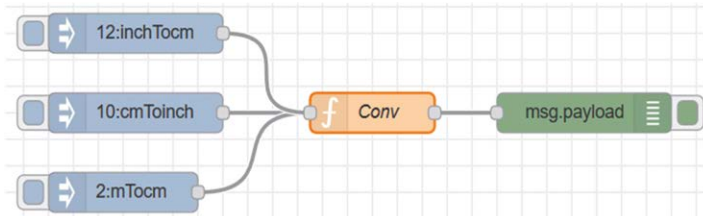


Figure 4.31 Join the 5 nodes together

- Click one of the **inject** nodes buttons to display the conversion in the Debug window as shown in Figure 4.32. In this figure, the inchToCm and mToCm nodes are clicked.



Figure 4.32 The Debug window

4.14 Multiple inputs and multiple outputs from a function

In this flow program, we will see how multiple inputs and multiple outputs can be handled in a function. In this example we have 3 inject nodes, a 3 input and 3 output function node, and debug node. The operation of this example is as follows:

When the button of **inject** node First is clicked, the message **First node clicked** is displayed by debug node Debug1

When the button of **inject** node Second is clicked, the message **Second node clicked** is displayed by debug node Debug2

When the button of **inject** node Third is clicked, the message **Third node clicked** is displayed by debug node Debug3

The steps are as follows:

- Create 3 **inject** nodes with the following characteristics:

Node	Topic
1	First
2	Second
3	Third

- Create a **function** node named **Func**, set its Outputs to 3, and enter the following code inside the function (see Figure 4.33). Notice that in a statement involving several **if** statements, it might be more readable to use a **switch** statement. Now click **Done**:

```
var conv = msg.topic

if(conv == "First")
{
    var msg1 = {payload: "First node clicked"};
    return [msg1,null,null];
}
else if(conv == "Second")
{
    var msg2 = {payload: "Second node clicked"};
    return [null,msg2,null];
}
else if(conv == "Third")
{
    var msg3 = {payload: "Third node clicked"};
    return [null,null,msg3];
}
```

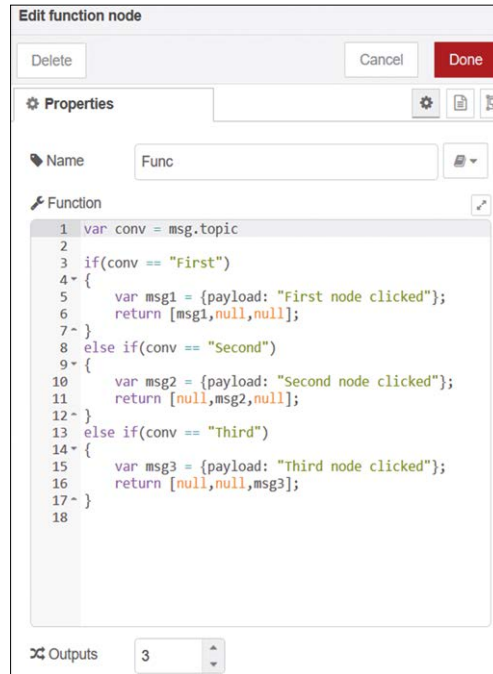


Figure 4.33 Create a function node

- Create 3 debug nodes with the following characteristics:

Node	Output	Name
1	msg.payload	Debug1
2	msg.payload	Debug2
3	msg.payload	Debug3

- Join all 7 nodes as shown in Figure 4.34, click Deploy

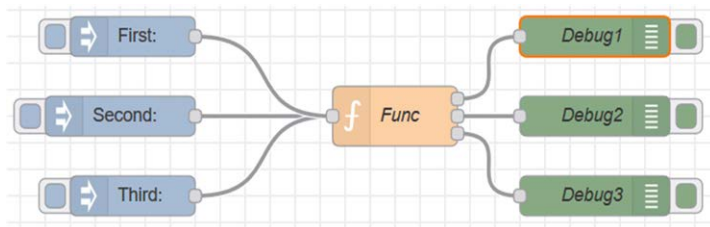


Figure 4.34 Join all the nodes

- Activate all 3 debug nodes
- Click one of the **inject** nodes buttons to display a message. Figure 4.35 shows the messages displayed in the Debug window when **inject** nodes First and Third were clicked.

```

04/12/2019, 14:48:19 node: Debug1
msg.payload : string[18]

"First node clicked"

04/12/2019, 14:48:21 node: Debug3
msg.payload : string[18]

"Third node clicked"

```

Figure 4.35 Debug window

4.15 Project 7 – Mean of numbers - Using the smooth node

In this project, we calculate the average of 4 numbers and then display the result in the Debug window. The flow program consists of 4 **inject** nodes, one **smooth** node, and one **debug** node as shown in Figure 4.36. The aim of this project is to show how to use the **smooth** node to calculate the mean of a group of numbers. In this project, the numbers whose mean is calculated are 3, 7, 8, and 2.

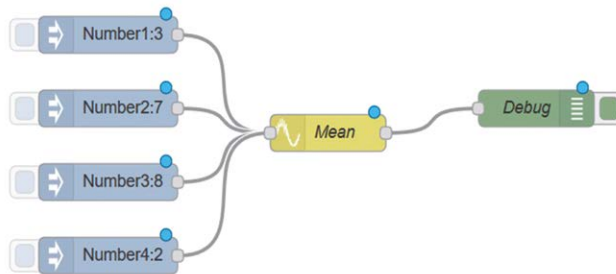


Figure 4.36 The flow program of Project 7

The steps are as follows:

- Create 4 **inject** nodes with the following characteristics:

Node	Payload	Topic
1	3	Number1
2	7	Number2
3	8	Number3
4	2	Number4

- Create a smooth node named **Mean**, as shown in Figure 4.37, click **Done**, and set the following parameters:

Property: msg.payload
 Action: Return the mean value
 Over the last 4 values
 Treat: All msg as one stream

Reduce: only emit one message per most recent N values
 Name: Mean

The Action is set to return the mean (average) of 4 input numbers. Treat is configured so that all the 4 input numbers are treated as one stream. The Reduce parameter is set to output only the calculated mean value and not the intermediate values.

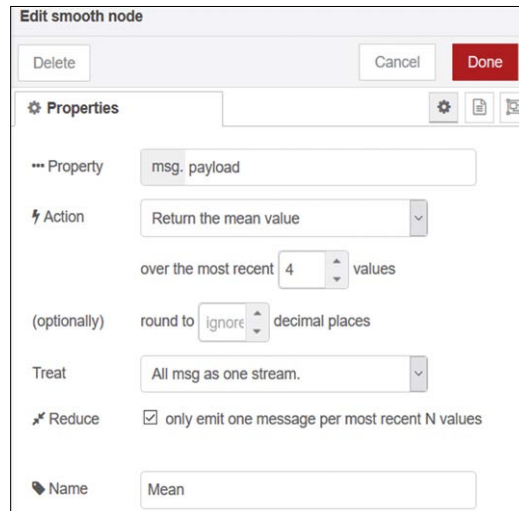


Figure 4.37 Create a smooth node

- Create a **debug** node named **Debug** and activate it.
- Click **Deploy** and make sure there are no error messages
- Click the buttons of the four **inject** nodes, starting from the top node. You will notice that no output is produced until the last inject node is clicked. The result is $(3+7+8+2) / 4 = 5$ is displayed in the Debug window as shown in Figure 4.38.

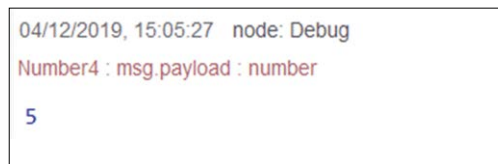


Figure 4.38 Displaying the mean value of numbers

4.16 Project 8 – Squares of numbers

In this project, we calculate and display the squares of numbers from 1 to 5 in the Debug window. The flow program in this project consists of 3 nodes: an **inject** node, a **function node**, and a **debug** node as shown in Figure 4.39.



Figure 4.39 The flow program of Project 8

The steps are as follows:

- Create an **inject** node with the payload called **Click**
- Create a **function** node named **Squares** and enter the following statements inside this function (Figure 4.40). The function displays the heading **SQUARES OF NUMBERS** and then calculates and displays the squares of numbers from 1 to 5:

```
var k;
var m;

for(k=0; k <= 5; k++)
{
    if(k == 0)
    {
        msg.payload = "SQUARES OF NUMBERS";
    }
    else
    {
        msg.payload="";
        msg.payload = msg.payload + "N= "+k + "    N*N= "+k*k;
    }
    node.send(msg);
}
return null;
```

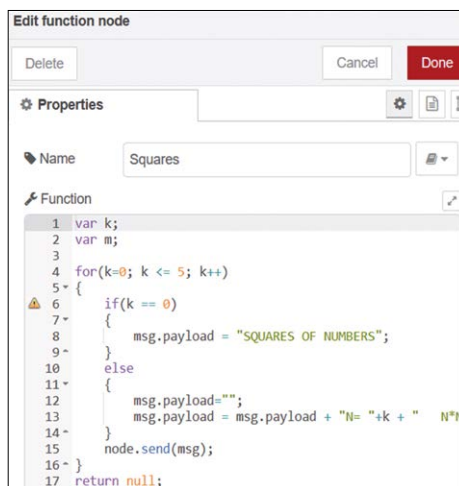


Figure 4.40 Function contents

- Create a debug node named **Debug**. Join all 3 nodes and click **Deploy**
- Click the inject nodes button and you should see the squares of numbers displayed as shown in Figure 4.41.

05/12/2019, 09:21:04	node: Debug
msg.payload : string[18]	
"SQUARES OF NUMBERS"	
05/12/2019, 09:21:04	node: Debug
msg.payload : string[13]	
"N= 1 N*N= 1"	
05/12/2019, 09:21:04	node: Debug
msg.payload : string[13]	
"N= 2 N*N= 4"	
05/12/2019, 09:21:04	node: Debug
msg.payload : string[13]	
"N= 3 N*N= 9"	
05/12/2019, 09:21:04	node: Debug
msg.payload : string[14]	
"N= 4 N*N= 16"	
05/12/2019, 09:21:04	node: Debug
msg.payload : string[14]	
"N= 5 N*N= 25"	

Figure 4.41 Displaying the squares of numbers

In this flow program, the **send** statement is used to output messages to the output of the function independent of the return values. i.e. output is not returned by the function, Here the function returns a null but the actual data from the function is output using the send statement.

4.17 Project 9 – Getting the weather reports – Display the local weather report

In this project, we will get the local weather report and display it in the Debug window. First of all, we have to install a node called **openweathermap**. The steps to install this node are as follows:

- Click **Menu -> Manage Palette**
- Click the **Install** tab and enter the following in the search path (see Figure 4.42)

node-red-node-openweathermap

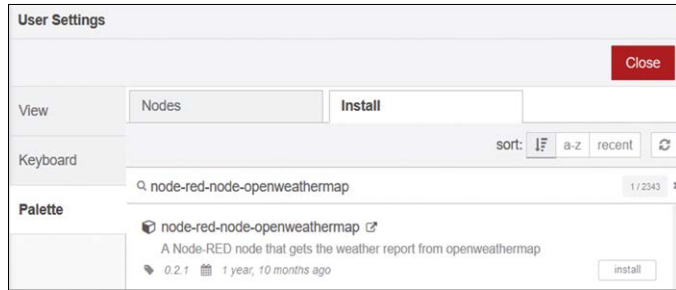


Figure 4.42 Search for the openweathermap

- Click **install** inside the right-hand side of the box displaying **node-red-node-openweathermap**
- Click **Close**

You should now see nodes **openweathermap** and **openweathermap in** displayed in your Node Palette in the Node-RED screen. If there is a problem, you can install the node **openweathermap** manually by entering the following command in the root directory of your Node-RED:

```
npm install node-red-node-openweathermap
```

Now that the **openweathermap** node is installed, we have to register and get an API key before we can use this node to get the local weather reports. An API key can be obtained as follows:

- Go to web site: <https://openweathermap.org/appid>
- Click the **API** tab at the top of the screen
- Click **Sign Up** and fill in the form to create a new account
- A new API key will be sent to the email address that you provided

We are now ready to create and use our flow program. The steps are given below:

- Click, drag and drop an **inject** node
- Click, drag, and drop the **openweathermap** node to the workspace
- Double click on the node and enter your API key, select the current weather and enter your city and country as shown in Figure 4.43. Name the node as **Local Weather**. Click **Done**

Figure 4.43 Configure node openweathermap

- Click, drag, and drop a debug node. Join the three nodes as shown in Figure 4.44.
Click Deploy



Figure 4.44 Join the two nodes

- Click **inject** nodes button. You should see the JASON formatted local weather report displayed in the Debug window as shown in Figure 4.45

```
05/12/2019, 12:24:30 node: e9103ab4.83ce8
msg.payload: Object
{
  "object": {
    "weather": "Clouds",
    "detail": "overcast clouds",
    "icon": "04d",
    "tempk": 279.25,
    "tempc": 6.1,
    "temp_maxc": 7.7,
    "temp_minc": 4.4,
    "humidity": 87,
    "pressure": 1020,
    "maxtemp": 280.93,
    "mintemp": 277.59,
    "windspeed": 3.1,
    "winddirection": 250,
    "location": "London",
    "sunrise": 1575532133,
    "sunset": 1575561206,
    "clouds": 99,
    "description": "The weather in London at coordinates: 51.51, -0.13 is Clouds (overcast clouds)."

```

Figure 4.45 JASON formatted weather report

As you can see, the report is very comprehensive, it shows that the weather was cloudy, the temperature at the time of making the request was 6.1°C, the humidity was 87%, atmospheric pressure was 1020 mbars, the wind speed was 3.1m/s and so on. The city where this report is generated for and its co-ordinates are given at the bottom part of the report. In the next project, we will add a function node and see how we can display only the current temperature in the Debug window.

4.18 Project 10 – Display the current temperature

In this project, we will get the local weather report and display only the current temperature in the Debug window. Here, the following nodes are used: **inject** node to start the flow, **openweathermap** node to get the current weather report, **function** node to extract the current temperature, and **debug** node to display the current temperature in the Debug window.

The steps are as follows:

- Create an **inject** node and an **openweathermap** node as in the previous project
- Create a **function** node and name it as **Temperature**. Double click this node and enter the following statements (see Figure 4.46):

```
var T;
T = msg.payload.tempc;
msg.payload = "Current temperature= "+T+ " Degrees C";
return msg;
```

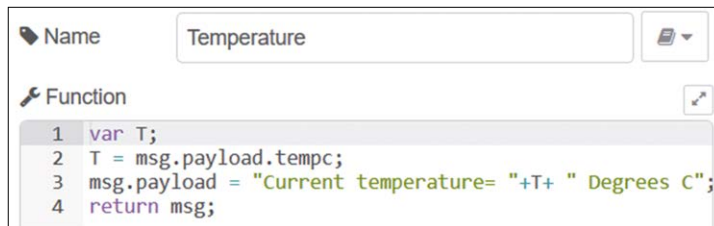


Figure 4.46 Function node contents

- Create a **debug** node. Join the 4 nodes together as in Figure 4.47. Click **Deploy**.

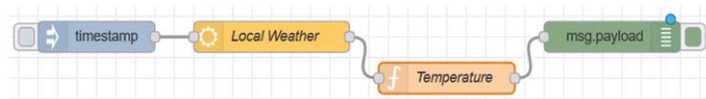


Figure 4.47 Join the 4 nodes together

- Click **inject** nodes button. You should see the current temperature displayed in the Debug window as shown in Figure 4.48

```
05/12/2019, 12:45:20 node: e9103ab4.83ce8
msg.payload : string[34]
"Current temperature= 6.2 Degrees C"
```

Figure 4.48 Displaying the current temperature

4.19 Project 11 – Sending the current temperature to an email

In this project, we will get the local weather report and send the current temperature to your email address. Here, the following nodes are used: **inject** node to start the flow, **openweathermap** node to get the current weather report, **function** node to extract the current temperature and the **email** node to send the email.

If the **email** node is not available in your Node Palette, you can install it as follows:

- Click **Menu -> Manage palette**
- Click the **Install** tab
- Enter the following and click install

```
node-red-node-email
```

- Close **Manage palette**. You should now have 2 email nodes in the Node-RED screen Node Palette

You can also install the email node by the following commands:

```
cd ~/.node-red
npm i node-red-node-email
```

Now that the email node is installed, we can carry on to create our flow program. The steps are given below:

- Create nodes: **inject**, **openweathermap**, and **function** exactly as in the previous project
- Click, drag and drop the **email** node (the one with an envelope picture on its right-hand side) to the workspace. Double click and configure as shown in Figure 4.49

Figure 4.49 Configure the email node

- Set the **To** field to the destination email address (you can specify more than one email addresses by separating them with commas), set the **Userid** to the email account from where the message will be sent from, and finally, set the password of this **Userid** account.
- Click **Done**
- Join the 4 nodes as shown in Figure 4.40 and click **Deploy**

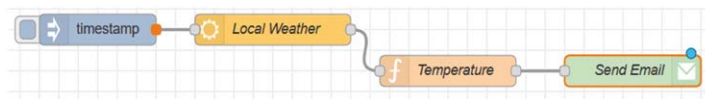


Figure 4.50 Join the 4 nodes

- If you are using a Gmail account, you will have to enable **less secure access** via your Google account settings. Alternatively, enter the following to your web browser:

`myaccount.google.com/u/1/lesssecureapps`

- Click the **inject** node button. The current temperature will be sent to the email address you specified. Figure 4.51 shows the mail sent to the author's email address.

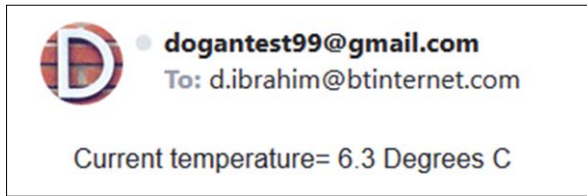


Figure 4.51 Received email

You can modify this project if you wish so that the temperature is sent at regular intervals to your email address. To do this, edit the inject node and set the repetition interval to the required value.

4.20 Project 12 – Sending the current temperature and atmospheric pressure to your twitter account

In this project, we will get the local weather report and send the current temperature and current atmospheric pressure readings to your Tweeter account. Here, the following nodes are used: **inject** node to start the flow, **openweathermap** node to get the current weather report, **function** node to extract the current temperature and the current atmospheric pressure, and the **twitter** node to tweet.

If the **twitter** node is not available in your Node Palette, you can install it as follows:

- Click **Menu -> Manage palette**
- Click the **Install** tab
- Enter the following and click install

```
node-red-node-twitter
```

- Close **Manage palette**. You should now have 2 twitter nodes in the Node-RED screen Node Palette: **twitter in** and **twitter out**

You can also install the twitter node using the following commands:

```
cd ~/.node-red
npm i node-red-node-twitter
```

Now that the twitter node is installed, we can carry on to create our flow program. The steps are given below:

- Create nodes: **inject**, and **openweathermap** exactly as in the previous project
- Create a **function** node (Figure 4.52), name it as **TempPress**, and enter the following statements inside this node:

```

var T;
T = msg.payload.tempc;
P = msg.payload.pressure;
msg.payload = "T= "+T+ " Degrees    P="+P+" mbars";
return msg;

```

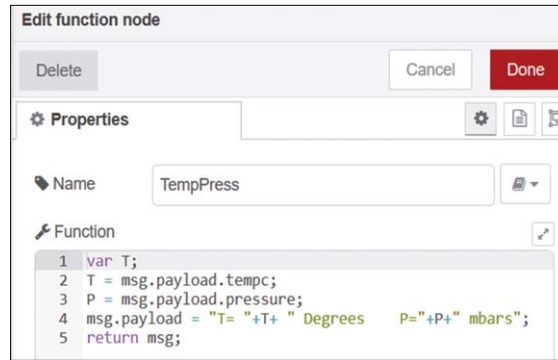


Figure 4.52 Contents of the function node

- Click, drag and drop a **twitter out** node and name it as **Tweeter**. Double click to configure this node. Enter your Twitter ID (e.g. @DoganIbrahim7) as shown in Figure 4.53

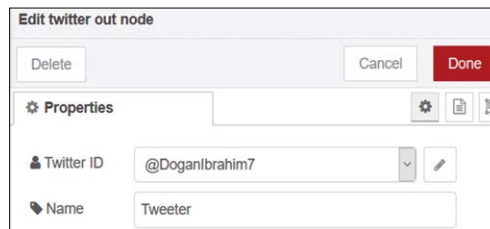
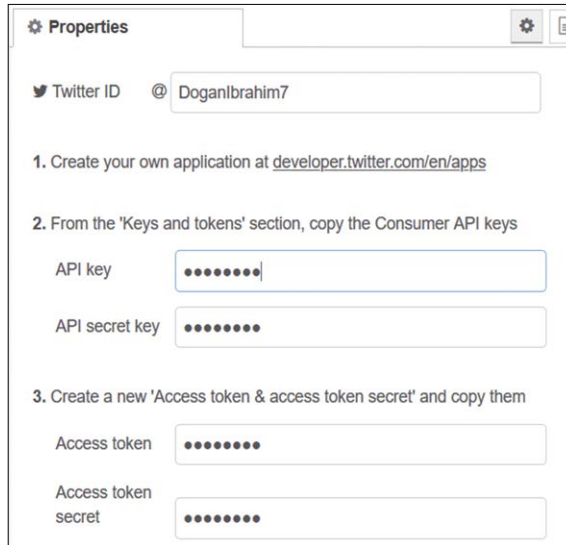


Figure 4.53 Enter your Twitter ID

- Click the pen symbol on the right-hand side of the Twitter ID. You will be presented with a screen where you have to enter your Twitter credentials. This requires registering at the Twitter site (developer.twitter.com/en/apps) and getting the following four keys unique to your application:

API Key, API secret Key, Access token, Access token secret

- Enter your Twitter ID and then copy and paste the four keys you obtained from the Twitter site (see Figure 4.54). Click **Done** to close the node data



Properties

Twitter ID @DoganIbrahim7

1. Create your own application at developer.twitter.com/en/apps

2. From the 'Keys and tokens' section, copy the Consumer API keys

API key

API secret key

3. Create a new 'Access token & access token secret' and copy them

Access token

Access token secret

Figure 4.54 Enter your credentials

- Now, join the 4 nodes as shown in Figure 4.55, and click **Deploy**
- Click the **inject** nodes button. The local temperature and atmospheric pressure readings will be sent to your twitter account as shown in Figure 4.55 which was copied from the author's mobile phone.



Figure 4.55 Temperature and atmospheric pressure sent to the twitter account

4.21 Node-RED configuration

Node-RED can be configured using a settings file. When Node-RED starts, it looks for a file called **settings.js** in the default user directory **~/.node-red**, and this file is loaded into the runtime as a **Node.js** module. It is also possible to define our settings file by using the option **-settings** in the command line.

The default settings file has most of its lines commented out. You can enable the commented lines but it is recommended that you know exactly what you are doing before enabling the commented lines. To enable a comment line, just remove the two slash characters (//) in front of the required line.

Additionally, there is a **Settings** option in the **Menu** when Node-RED is up and running,

where the user **View**, **Keyboard**, and **Palette** options can be configured as shown in Figure 4.56.

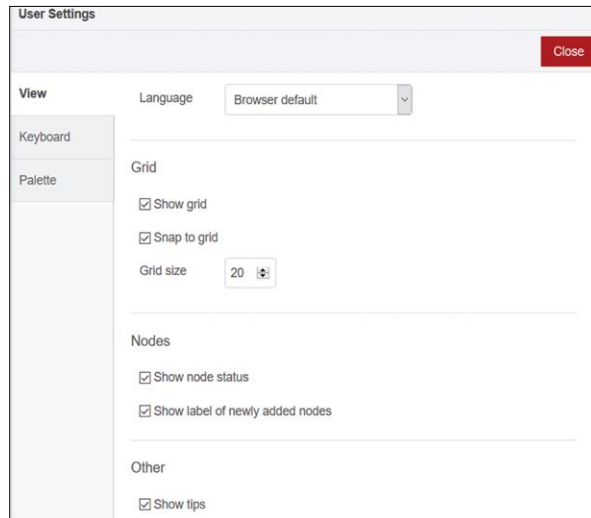


Figure 4.56 Settings menu option

4.22 Summary

In this chapter, we have learned how to start Node-RED software on Raspberry Pi. In addition, we have learned how to use some Node-RED nodes in various flow programs. Several examples and projects are given in this chapter to make the reader familiar with some of the important concepts of Node-RED.

In the next chapter, we will be learning how to use Node-RED to access Raspberry Pi GPIO pins, and how to interface sensors and actuators to Raspberry Pi, and how to control them from Node-RED based flow programs.

Chapter 5 • Node-RED based Raspberry Pi projects using GPIO

5.1 Overview

All versions of Raspberry Pi include a socket for connecting external devices, such as sensors, LEDs, actuators, etc. In this chapter, we will look at how external devices connected to Raspberry Pi can be accessed from Node-RED flow programs, by developing projects using Node-RED.

All the projects given in this chapter will include the following sections to make it easy for the reader to understand how these projects are developed:

- Title
- Description of the project
- Aim
- Background information (where necessary)
- Block diagram (where necessary)
- Circuit diagram
- Node-RED flow program with a full description
- Suggestions for additional work (where necessary)

Before going into the details of the projects, it is worthwhile to look at the Raspberry Pi GPIO connector. This is a 40-pin dual-in-line 2.54mm wide connector as shown in Figure 5.1.

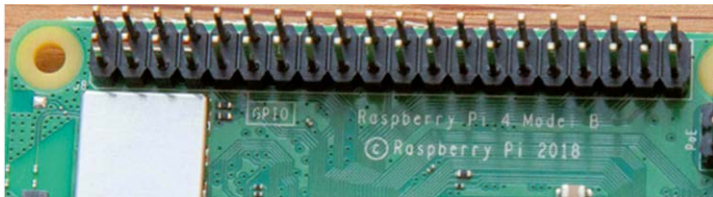


Figure 5.1 Raspberry Pi 4 GPIO connector

5.2 GPIO – Parallel interface

When the GPIO connector is at the far side of the board, the pins at the bottom, starting from the left of the connector are numbered as 1, 3, 5, 7, and so on, while the ones at the right of the connector are numbered as 2, 4, 6, 8 and so on (see Figure5.2).

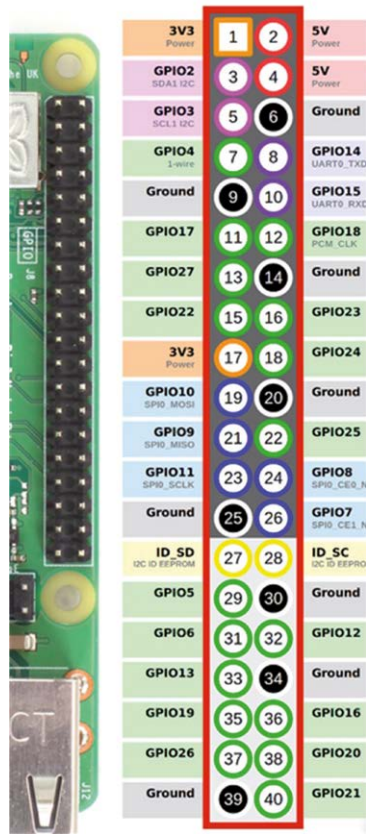


Figure 5.2 Raspberry Pi GPIO pin configuration

The GPIO provides 26 general purpose bi-directional I/O pins. Some of the pins have multiple functions. For example, pins 3 and 5 are the GPIO2 and GPIO3 input-output pins respectively. These pins can also be used as the I2C bus I2C SDA and I2C SCL pins respectively. Similarly, pins 9,10,11,19 can either be used as general-purpose input-output, or as the SPI bus pins. Pins 8 and 10 are reserved for UART serial communication.

Two power outputs are provided: +3.3V and +5.0V. The GPIO pins operate at +3.3V logic levels (not like many other computer circuits that operate with +5V). A pin can either be an input or an output. When configured as an output, the pin voltage is either 0V (logic 0) or +3.3V (logic 1). Raspberry Pi is normally operated using an external power supply (e.g. a mains adapter) with +5V output and minimum 2A current capacity. A 3.3V output pin can supply up to 16mA of current. The total current drawn from all output pins should not exceed the 51mA limit. Care should be taken when connecting external devices to the GPIO pins as drawing excessive currents or short-circuiting a pin can easily damage your Raspberry Pi. The amount of current that can be supplied by the 5V pin depends on many factors such as the current required by the Pi itself, current taken by the USB peripherals, camera current, HDMI port current, and so on.

When configured as an input, a voltage above +1.7V will be taken as logic 1, and a voltage below +1.7V will be taken as logic 0. Care should be taken not to supply voltages greater than +3.3V to any I/O pin as large voltages can easily damage your Pi.

5.3 Project 13 – LED control

Description: In this project, we will connect an LED to one of the Raspberry Pi GPIO pins and then turn the LED ON or OFF from a Node-RED flow program.

Aim: The aim of this project is to show how an LED can be connected to a GPIO port pin and how it can be controlled from a Node-RED flow program.

Background information: The forward voltage of a typical LED is around 1.8V. The current through the LED depends on the required light intensity and the type of LED used. In general, 3mA should give enough visible light for small LEDs. Because the output voltage of a GPIO pin is +3.3V we have to use a current limiting resistor in series with the LED. The value of the current limiting resistor is calculated as:

$$R = (3.3V - 1.8V) / 3mA = 500 \text{ Ohm. We can choose a } 470 \text{ Ohm resistor}$$

Be careful that the LED has two pins: anode (long pin) and cathode (short pin). The cathode pin must be connected to ground (see Figure 5.3).

Circuit diagram: Figure 5.4 shows the circuit diagram of the project where the anode of the LED is connected to pin GPIO 2 of the Raspberry Pi through a current limiting resistor.

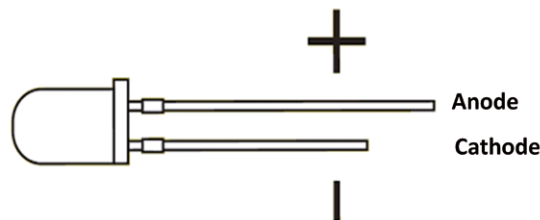


Figure 5.3 Pins of an LED

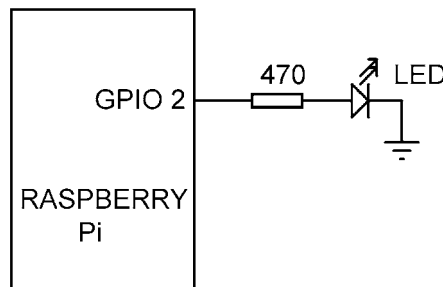


Figure 5.4 LED connected to pin GPIO 2

Node-RED flow program: Figure 5.5 shows the flow program. In this program, two **inject** nodes and a **rpi gpio out** node are used.

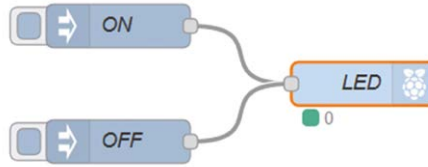


Figure 5.5 Flow program of the project

The steps are as follows:

- Create an **inject** node with the configuration as shown in Figure 5.6

The image shows the configuration window for an inject node in Node-RED. The window has a title bar with a gear icon and a close button. The main area contains several fields: 'Payload' is set to '1', 'Topic' is empty, 'Inject once after' is set to '0.1' seconds, 'Repeat' is set to 'none', and 'Name' is set to 'ON'.

Figure 5.6 Create an inject node

- Create another **inject** node with the configuration as shown in Figure 5.7

The image shows the configuration window for an inject node in Node-RED. The window has a title bar with a gear icon and a close button. The main area contains several fields: 'Payload' is set to '0', 'Topic' is empty, 'Inject once after' is set to '0.1' seconds, 'Repeat' is set to 'none', and 'Name' is set to 'OFF'.

Figure 5.7 Create another inject node

- Create a **rpi gpio out** node with the configuration as shown in Figure 5.8, set the Type to Digital output, Initialize pin state to 0, and name it as LED. Click **Done**

Pin

3.3V Power - 1

2 - 5V Power

SDA1 - GPIO02 - 3

4 - 5V Power

SCL1 - GPIO03 - 5

6 - Ground

GPIO04 - 7

8 - GPIO14 - TxD

Ground - 9

10 - GPIO15 - RxD

GPIO17 - 11

12 - GPIO18

GPIO27 - 13

14 - Ground

GPIO22 - 15

16 - GPIO23

3.3V Power - 17

18 - GPIO24

MOSI - GPIO10 - 19

20 - Ground

MISO - GPIO09 - 21

22 - GPIO25

SCLK - GPIO11 - 23

24 - GPIO8 - CE0

Ground - 25

26 - GPIO7 - CE1

SD - 27

28 - SC

GPIO05 - 29

30 - Ground

GPIO06 - 31

32 - GPIO12

GPIO13 - 33

34 - Ground

GPIO19 - 35

36 - GPIO16

GPIO26 - 37

38 - GPIO20

Ground - 39

40 - GPIO21

Type

Digital output

Figure 5.8 Create a rpi gpio out node

- Join the 3 nodes together as shown in Figure 5.5, and click **Deploy**.
- Connect the LED to Raspberry Pi as shown in Figure 5.4. The circuit was built on a breadboard and connected to Raspberry Pi GPIO connector using jumper wires as shown in Figure 5.9.

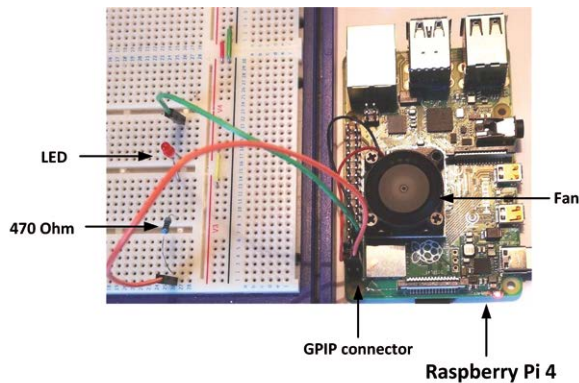


Figure 5.9 Circuit constructed on a breadboard

- Click the button of **inject** node ON, you should see the LED turning ON. Click the button of **inject** node OFF, and this time the LED should turn OFF

As shown in Figure 5.10, you can, if you wish, connect **debug** nodes to the flow so that the state of the LED can be displayed at any time in the Debug window. Figure 5.11 shows the LED state as the LED is turned ON or OFF.

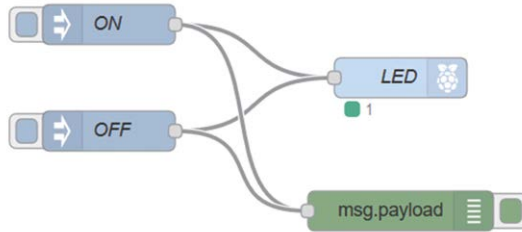


Figure 5.10 Connecting debug nodes to the flow



Figure 5.11 State of the LED displayed

5.4 Project 14 – Flashing LED

Description: In this project, we will connect an LED to one of the Raspberry Pi GPIO pins as in the previous project, and then flash the LED every second.

Aim: The aim of this project is to show how an LED connected to a GPIO port pin can be flashed using a **trigger** node.

Circuit diagram: The LED is connected to Raspberry Pi as in Figure 5.4.

Node-RED flow program: Figure 5.12 shows the flow program. In this program an **inject** node, a **trigger** node, and a **rpi gpio out** node are used.



Figure 5.12 Flow program of the project

The steps are as follows:

- Create an **inject** node which runs regularly every 2 seconds, whose configuration is shown in Figure 5.13. Click **Done**.

The screenshot shows the configuration for an Inject node. It includes a 'Payload' dropdown set to 'a_z', an empty 'Topic' field, an unchecked checkbox for 'Inject once after 0.1 seconds, then', a 'Repeat' section with 'interval' selected, 'every 2 seconds', and an empty 'Name' field.

Figure 5.13 Create an inject node

- Create a **trigger** node to send out 1, wait for 1 second and then send out 0, with the configuration as shown in Figure 5.14. Click **Done**.

The screenshot shows the configuration for a Trigger node. It includes a 'Send' dropdown set to 'a_z 1', a 'then' dropdown set to 'wait for', a '1 Seconds' delay, an unchecked checkbox for 'extend delay if new message arrives', a 'then send' dropdown set to 'a_z 0', a 'Reset the trigger if:' section with 'msg.reset is set' and 'msg.payload equals optional', and a 'Handling' dropdown set to 'all messages'.

Figure 5.14 Create a trigger node

- Create the **rpi gpio out** node as in the previous project. Join the 3 nodes, and click **Deploy**.
- Click the **inject** node's button. You should see the LED flashing every second.

In this project, the **inject** node activates the **trigger** node every two seconds, which in turn controls the LED by turning it ON for a second, and OFF for a second.

You could insert a **debug** node to the flow program (Figure 5.15) so that the state of the LED can be displayed. Additionally, you can see the timing accuracy of the project in the Debug window. This is shown in Figure 5.16.

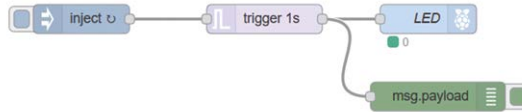


Figure 5.15 Inserting a debug node

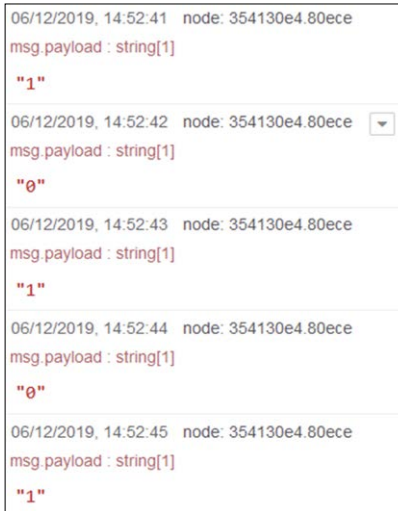


Figure 5.16 Displaying the state of the LED

5.4.1 Using a context variable to flash the LED

In the previous program, we used a **trigger** node together with an **inject** node and a **rpi gpio out** node to flash the LED. In this section, we will learn how to use a **function** node instead of the **trigger** node to flash the LED.

The function in this example will make use of context variables. A context variable is a static storage variable in a node that does not lose its value between calls.

Figure 5.17 shows the flow program using a function. In this example, the **inject** node is given the name **Every second** and it is configured to repeat every second. The **function** node is given the name **Toggle LED** and its contents are as follows:

```

if(context.led == 0)
    context.led = 1;
else
    context.led = 0;
msg.payload = context.led;
return msg;
  
```



Figure 5.17 Flow program of the project

5.5 Project 15 – Alternately flashing LEDs

Description: In this project, we will connect two LEDs to our Raspberry Pi GPIO port pins and then flash these LEDs alternately every second.

Aim: The aim of this project is to show how more than one LED connected to GPIO port pins and also how these LEDs can be flashed alternately using **trigger** nodes.

Circuit diagram: The LEDs are connected to Raspberry Pi port pins GPIO 2 and GPIO 3 as shown in Figure 5.18.

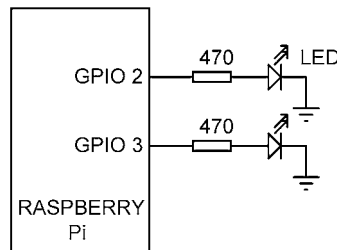


Figure 5.18 Circuit diagram of the project

Node-RED flow program: Figure 5.19 shows the flow program. In this program an **inject** node, two **trigger** nodes, and two **rpi gpio out** node are used, one for each LED.

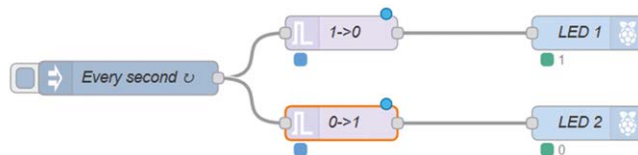


Figure 5.19 Flow program of the project

The steps are as follows:

- Create an **inject** node which runs regularly every 2 seconds
- Create a **trigger** node to send out 1, wait for 1 second and then send out 0, with the configuration as shown in Figure 5.14, name it as 1->0, and click **Done**.
- Create another **trigger** node to send out 0, wait for 1 second and then send out 1, name it as 0->1, and click **Done**.
- Create a **rpi gpio out** node as in the previous project and set the GPIO pin number to 2
- Create another **rpi gpio out** node as before, and set the GPIO pin number to 3

- Click **Deploy**
- Click the **inject** node's button. You should now see the two LEDs flashing alternately every second.

In this project, the **trigger** node 1->0 outputs a falling signal level, while node 0->1 outputs a rising signal level. The result is that when one LED is ON, the other one is OFF, and vice-versa. The **inject** node activates the **trigger** node every two seconds as before.

As in Project 13, you can create two **debug** nodes and attach to the outputs of the two **trigger** nodes to display the state of the two LEDs.

5.5.1 Using context variables to flash the LEDs alternately

In the previous program, we used **trigger** nodes together with an **inject** node and two **rpi gpio out** nodes to flash the two LEDs alternately. In this section, we will learn how to use a **function** node instead of the two **trigger** nodes to flash the LEDs.

The function in this example is configured to have two outputs, and it will make use of two context variables as was described in section 5.4.1.

Figure 5.20 shows the flow program using a function. In this example, the **inject** node is given the name **Every second** and it is configured to repeat every second.



Figure 5.20 Flow program of the project

The **function** node is given the name **Toggle LEDs** and its contents are as follows (see Figure 5.21):

```
if(context.led1 == 0)
{
    context.led1 = 1;
    context.led2 = 0;
}
else
{
    context.led1 = 0;
    context.led2 = 1;
}

var msg1 = {payload:context.led1};
var msg2 = {payload:context.led2};
return [msg1, msg2];
```

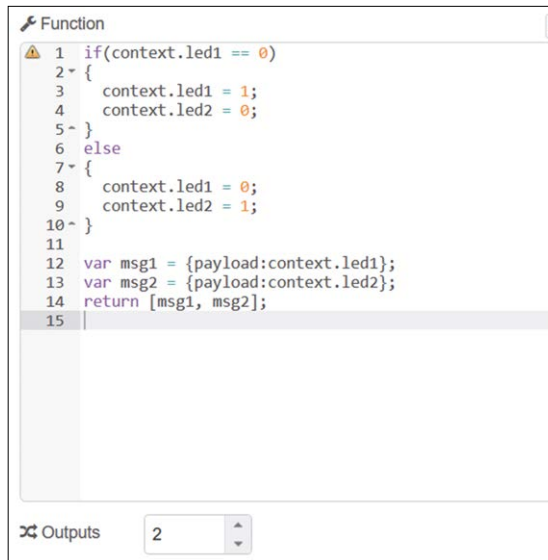


Figure 5.21 Function configuration

Click the **inject** node's button. You should see the two LEDs flashing alternately every second.

Notice how the LED data for each LED is sent to the corresponding outputs of the function in the return statement. Figure 5.22 shows the project constructed on a breadboard.

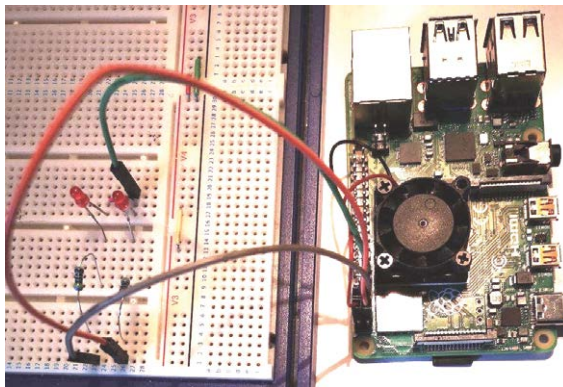


Figure 5.22 The project constructed on a breadboard

5.6 Project 16 – Temperature alarm with buzzer

Description: This is a simple temperature alarm system. A small active buzzer is connected to one of the GPIO ports of the Raspberry Pi. The local temperature is read from the weather site as in Project 9, and if it is above a pre-defined threshold value then the buzzer is activated. The threshold value is taken to be 30°C in this project, but it can be changed any other value if desired. The temperature is checked at every minute.

Aim: The aim of this project is to show how a simple temperature alarm system can be designed with Node-RED, using Raspberry Pi and a small buzzer as the hardware.

Background information: Small buzzers (also known as piezoelectric buzzers) are used commonly in electronic alarm projects. These buzzers are operated with any voltage from +3V to +5V. There are two types of buzzers: active buzzers, and passive buzzers. Active buzzers have built-in electronic circuits and they sound at a fixed frequency when a DC voltage is applied across their terminals. Usually, the sound frequency of these buzzers is around 1kHz. It is very easy to use the active buzzers since they can easily be connected to the output port pin of the Raspberry Pi. Passive buzzers, on the other hand, require a waveform (usually PWM or sinewave) to be applied across their terminals, and the frequency of the generated sound depends on the frequency of the applied waveform. Although it may be more difficult to use passive buzzers, they have the advantages that the sound frequency can easily be changed, for example, they can be used in simple electronic organ projects to generate sounds at different frequencies. In this project, an active buzzer is used.

Block diagram: Figure 5.23 shows the block diagram of the project.

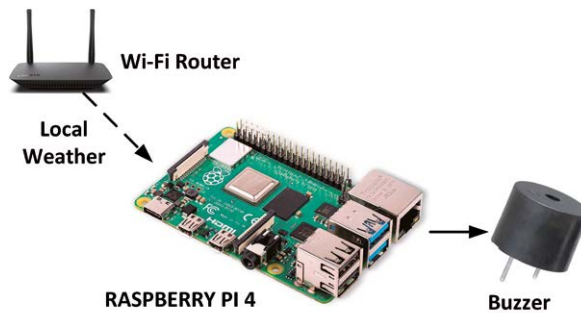


Figure 5.23 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 5.24. The buzzer is connected directly to port pin GPIO 2 of the Raspberry Pi 4.

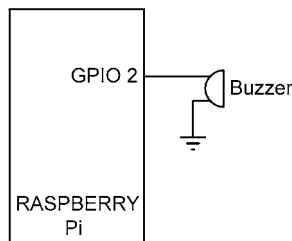


Figure 5.24 Circuit diagram of the project

Node-RED flow program: Figure 5.25 shows the flow program. In this program 4 nodes are used: an **inject** node that repeats every minute, an **openweathermap** node that gets the local weather report, a **function** node that extracts the current temperature from the

weather report and sends out 1 to the **rpi gpio out** node to activate the buzzer if the temperature is greater than 30°C, otherwise, 0 is sent to the **rpi gpio out** node to deactivate the buzzer.

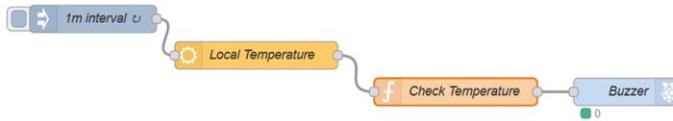


Figure 5.25 Flow program of the project

The steps are as follows:

- Create an **inject** node with the name **1m interval**, and set the repeat interval to 1 minute
- Create an **openweathermap** node with the name **Local Temperature**, enter the API key (see Project 9), and set your local city name
- Create a **function** node with the name **Check Temperature**, and enter the following statements into this node (see Figure 5.26). Here, if the temperature is greater than 30°C then the buzzer is activated, otherwise, the buzzer is deactivated:

```
var T;
T = msg.payload.tempc;
if(T > 30)
    msg.payload = 1;
else
    msg.payload = 0;
return msg;
```

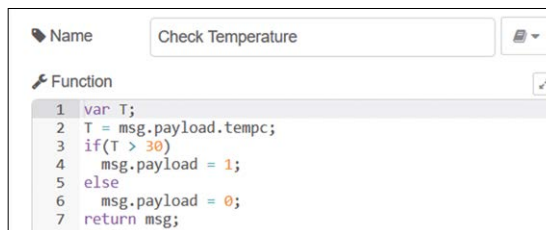


Figure 5.26 Configure the function node

- Create a **rpi gpio out** node named **Buzzer**, and set the GPIO pin to 2
- Join all 4 nodes as in Figure 5.25, and click **Deploy**
- Click the **inject** node's button. The local temperature report will now be obtained every minute and if it is greater than 30°C then the buzzer will sound

You can modify the hardware easily, for example by replacing the buzzer with a relay so that a heater can be turned ON or OFF, or an LED can be used to indicate visually when an alarm condition occurs. Additionally, a **debug** node can be connected to the output of the **function** node to display the status of the buzzer (1 or 0 will be displayed in the Debug window)

5.7 Project 17 – Controlling a GPIO port remotely using email

Description: In this project, we will control a Raspberry Pi GPIO port remotely by sending an email. A small active buzzer is connected to one of the GPIO ports of the Raspberry Pi. The buzzer is activated by sending the command ON by email. Similarly, the buzzer is deactivated by sending the command OF by email.

Aim: The aim of this project is to show how a Raspberry Pi GPIO port can be controlled remotely by sending an email.

Block diagram: The block diagram of the project is as shown in Figure 5.23.

Circuit diagram: The circuit diagram of the project is as shown in Figure 5.24, where the buzzer is connected directly to port pin GPIO 2 of the Raspberry Pi 4.

Node-RED flow program: Figure 5.27 shows the flow program. In this program 3 nodes are used: an **email in** node that gets emails at regular intervals, a **function** node that sends 1 or 0 to its output depending on whether the user command is ON or OF respectively, and a **rpi gpio out** node to control the buzzer.



Figure 5.27 Flow program of the project

The steps are as follows:

- Click, drag and drop the **email in** node to the workspace. Configure this node as shown in Figure 5.28 and click **Done**. The node is configured to check for incoming emails every 10 seconds. The emails are expected to come to the Gmail account with username: dogantest99@gmail.com, where the password should also be specified (you will have to replace these to your own Gmail account details)

The screenshot shows the 'Properties' panel for a 'Get mail' node in Node-RED. The configuration is as follows:

- Get mail:** automatically (dropdown), every 10 seconds
- Protocol:** IMAP (dropdown)
- Use SSL?:** ☒
- Server:** imap.gmail.com
- Port:** 993
- Userid:** dogantest99@gmail.com
- Password:** [masked with dots]
- Folder:** INBOX
- Disposition:** Mark Read (dropdown)
- Criteria:** Unseen (dropdown)
- Name:** Name

Figure 5.28 Configure the email in node

- Create a function node named **ON/OFF**, having the following statements (see Figure 5.29). Here, function **substr** is used to extract the first two characters of the received email command. If the command is **ON**, then 1 is returned to activate the buzzer. Otherwise, if the command is **OF**, then 0 is returned to deactivate the buzzer. Click **Done**:

```
var cmd = msg.payload.substr(0,2);
var On = {payload: "1"};
var Off = {payload: "0"};

if(cmd == "ON")
    return On;
else if(cmd == "OF")
    return Off;
```

The screenshot shows the 'Properties' panel for a 'Function' node in Node-RED. The configuration is as follows:

- Name:** ON/OFF
- Function:**

```
1 var cmd = msg.payload.substr(0,2);
2 if(cmd == "ON")
3     msg.payload = 1;
4 else if(cmd == "OF")
5     msg.payload = 0;
6 return msg;
```

Figure 5.29 Configuration of the function node

- Create a **rpi gpio out** node and set the port number to GPIO 2. Join the 3 nodes and click **Deploy**

The program now should check the email account every 10 seconds. You should see the words **connecting** and **fetching** displayed under the **email in** node just before the account is checked. In this project, the buzzer should sound when an email is sent to account dogantest@gmail.com with the mail content having the word **ON**. Similarly, the buzzer should stop sounding when the word **OF** (note it is OF and not OFF) is sent by email.

Just a reminder, if you are using a Gmail account, you will have to enable **less secure access** via your Google account settings. Alternatively, enter the following to your web browser:

myaccount.google.com/u/1/lesssecureapps

The buzzer in this project can be replaced for example with a relay if desired so that electrical equipment (e.g. home appliances) can be controlled remotely by sending emails from anywhere in the world.

5.8 Project 18 – Confirmation of the buzzer state

Description: In the previous project we have seen how to control a Raspberry Pi GPIO port remotely by sending emails. The problem here is that when we send a command to the Raspberry Pi we are not sure whether or not the GPIO port has taken the required value. In this project, we read the state of the GPIO port and send an email to confirm the state of the port pin. The confirmation is very important in some applications, for example, if we wish to activate our home boiler using a relay, we want to make sure that the boiler has actually been activated.

If the command **ON** is sent to activate the buzzer, then the message **Buzzer is ON** will be sent to the specified email address when the buzzer is actually turned ON. If on the other hand command **OF** is sent, then the message **Buzzer is OFF** will be sent to the specified email address.

Aim: The aim of this project is to show how the state of a GPIO port pin can be read and then a message can be sent to an email account to confirm the state of the GPIO port pin.

Block diagram: The block diagram of the project is as shown in Figure 5.23.

Circuit diagram: The circuit diagram of the project is as shown in Figure 5.30. Here, the buzzer is connected directly to port pin GPIO 2 of the Raspberry Pi 4 as in the previous project, and GPIO 2 is connected to GPIO 3 where GPIO 3 is configured as an input in Node-RED. The state of the buzzer will be read by GPIO 3 and a message will be sent to an email account.

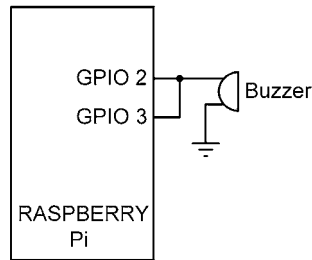


Figure 5.30 Circuit diagram of the project

Node-RED flow program: Figure 5.31 shows the flow program. In this program 6 nodes are used: 3 nodes to receive the email and control the buzzer accordingly, and 3 nodes to send the state of the buzzer port to the specified email address.

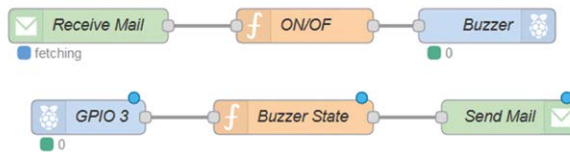


Figure 5.31 Flow program of the project

The steps are as follows:

- The first 3 nodes used to receive emails is same as in the previous project and are not repeated here again (only the name of the email in node is changed)
- The next 3 nodes are used to receive the state of port GPIO 3 and send an email to the specified address.
- Click **rpi gpio in** node, drag and drop it to the workspace. Name this node as **GPIO 3** and set it to receive the state of port GPIO 3 as shown in Figure 5.32, click **Done**. This port will receive the state of port pin GPIO 2 which is connected to GPIO 3.

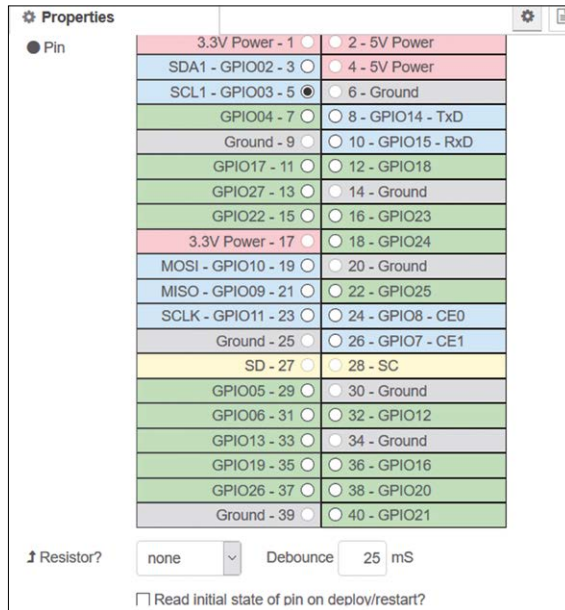


Figure 5.32 Configure the rpi gpio in node

- Create a **function** node, name it as **Buzzer State**, enter the following statements, and click **Done** (Figure 5.33):

```
if(msg.payload == "1")
    msg.payload = "Buzzer is ON"
else
    msg.payload = "Buzzer is OFF"
return msg;
```

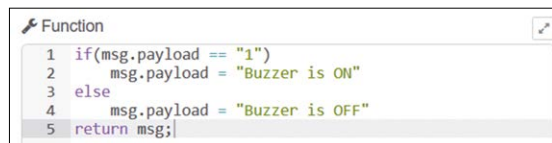


Figure 5.33 Configure the function node

- Click on **email out** node, drag and drop it in the workspace and name it as **Send Mail**. Configure this node as shown in Figure 5.34. The state of the buzzer is sent to email address: d.ibrahim@btinternet.com from email address: dogantest99@gmail.com (you should enter your own email addresses and passwords)

Figure 5.34 Configure the email out node

- Join the nodes as in Figure 5.31, and click **Deploy**. To test the program, send command **ON** to email address dogantest99@gmail.com, and the buzzer should be activated. At the same time, the message **Buzzer is ON** will be sent to email address d.brahim@btinternet.com (you should use your own email addresses and passwords).

5.9 Project 19 – Controlling multiple GPIO ports remotely using email

Description: In this project, we will control 4 Raspberry Pi GPIO ports remotely by sending an email. Four LEDs are connected to the Raspberry Pi GPIO ports through current limiting resistors. The LEDs are turned ON/OFF by sending email commands in the following form:

```
ON=1    turn ON LED1
ON=2    turn ON LED2
ON=3    turn ON LED3
ON=4    turn ON LED4
OF=1    turn OFF LED1
OF=2    turn OFF LED2
OF=3    turn OFF LED3
OF=4    turn OFF LED4
```

Aim: The aim of this project is to show how multiple GPIO ports of the Raspberry Pi can be controlled remotely by sending Email messages.

Block diagram: Figure 5.35 shows the block diagram of the project.

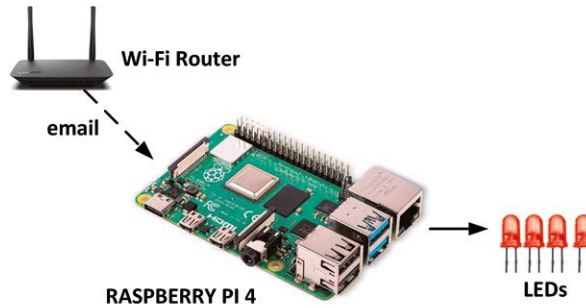


Figure 5.35 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 5.36. The LEDs are connected to the following Raspberry Pi port pins through 470 Ohm current limiting resistors:

LED	Raspberry Pi Port Name	Raspberry Pi Port Pin Number
LED1	GPIO 2	3
LED2	GPIO 3	5
LED3	GPIO 4	7
LED4	GPIO 17	11

Port pin number 9 of the Raspberry Pi is the ground pin that is connected to the cathode pins of all the LEDs.

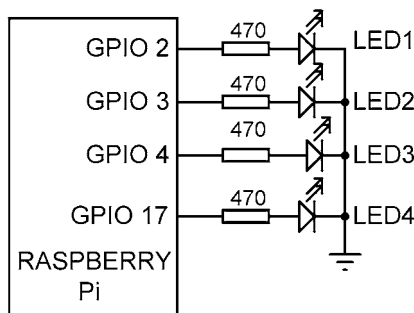


Figure 5.36 Circuit diagram of the project

Node-RED flow program: Figure 5.37 shows the flow program. In this program 6 nodes are used: an **email in** node to check and get the email at regular intervals, a **function** node with 4 outputs to decode the user commands, and 4 **rpi gpio out** nodes to control the 4 LEDs.

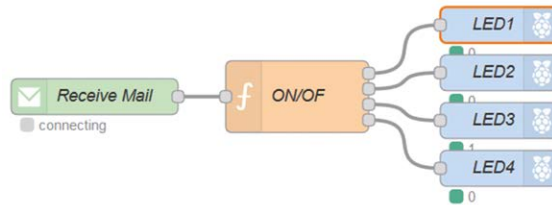


Figure 5.37 Flow program of the project

The steps are as follows:

- Create **email** in node as in the previous projects and name it as **Receive Email**
- Create a **function** node named **ON/OFF** with the following statements (see Figure 5.38). Variable **cmd** extracts the first 3 characters of the command and for valid commands, it has the value of **ON=** or **OF=**. Variable **port** extracts the required LED number and for valid commands, it has the values of **1**, **2**, **3**, or **4**. Notice how multiple outputs are handled in the function:

```
var cmd = msg.payload.substr(0,3);
var port = msg.payload.substr(3,1);
var On = {payload: "1"};
var Off = {payload: "0"};

if(cmd == "ON=")
{
    if(port == "1")
        return [On, null, null, null];
    else if(port == "2")
        return [null, On, null, null];
    else if(port == "3")
        return [null, null, On, null];
    else if(port == "4")
        return [null, null, null, On]
}

if(cmd == "OF=")
{
    if(port == "1")
        return [Off, null, null, null];
    else if(port == "2")
        return [null, Off, null, null];
    else if(port == "3")
        return [null, null, Off, null];
    else if(port == "4")
        return [null, null, null, Off]
}
```

```

1 var cmd = msg.payload.substr(0,3);
2 var port = msg.payload.substr(3,1);
3 var On = {payload: "1"};
4 var Off = {payload: "0"};
5
6 if(cmd == "ON=")
7 {
8   if(port == "1")
9     return [On,null,null,null];
10  else if(port == "2")
11    return [null,On,null,null];
12  else if(port == "3")
13    return [null,null,On,null];
14  else if(port == "4")
15    return [null,null,null,On];
16 }
17
18 if(cmd == "OF=")
19 {
20   if(port == "1")
21     return [Off,null,null,null];
22   else if(port == "2")
23     return [null,Off,null,null];
24   else if(port == "3")

```

Figure 5.38 Configuring the function (only part of the code is shown)

- Create 4 **rpi gpio out** nodes, one for each of the LEDs, LED1, LED2, LED3, and LED4. Set the port pin names to **GPIO 2**, **GPIO 3**, **GPIO 4** and **GPIO 17** respectively and clear the initial states of the ports to 0. An example for **GPIO 17** is shown in Figure 5.39. Click **Done** for each node.

SCL1 - GPIO03 - 5	<input type="radio"/>	6 - Ground
GPIO04 - 7	<input type="radio"/>	8 - GPIO14 - TxD
Ground - 9	<input type="radio"/>	10 - GPIO15 - RxD
GPIO17 - 11	<input checked="" type="radio"/>	12 - GPIO18
GPIO27 - 13	<input type="radio"/>	14 - Ground
GPIO22 - 15	<input type="radio"/>	16 - GPIO23
3.3V Power - 17	<input type="radio"/>	18 - GPIO24
MOSI - GPIO10 - 19	<input type="radio"/>	20 - Ground
MISO - GPIO09 - 21	<input type="radio"/>	22 - GPIO25
SCLK - GPIO11 - 23	<input type="radio"/>	24 - GPIO8 - CE0
Ground - 25	<input type="radio"/>	26 - GPIO7 - CE1
SD - 27	<input type="radio"/>	28 - SC
GPIO05 - 29	<input type="radio"/>	30 - Ground
GPIO06 - 31	<input type="radio"/>	32 - GPIO12
GPIO13 - 33	<input type="radio"/>	34 - Ground
GPIO19 - 35	<input type="radio"/>	36 - GPIO16
GPIO26 - 37	<input type="radio"/>	38 - GPIO20
Ground - 39	<input type="radio"/>	40 - GPIO21

Type:

☒ Initialise pin state?

initial level of pin - low (0)

Figure 5.39 Configuring the port for LED4

- Join the nodes as shown in Figure 5.37 and click **Deploy**. To test the program, connect the hardware as in Figure 5.36 and send an email for example with the contents of **ON=2**. LED2 should turn ON. Send an email with **OF=2** and this time LED2 should turn OFF

The LEDs used in this project can be replaced with relays if desired so that electrical equipment can easily be controlled remotely by sending emails.

Figure 5.40 shows the project constructed on a breadboard where the LEDs are connected to the Raspberry Pi using jumper wires.

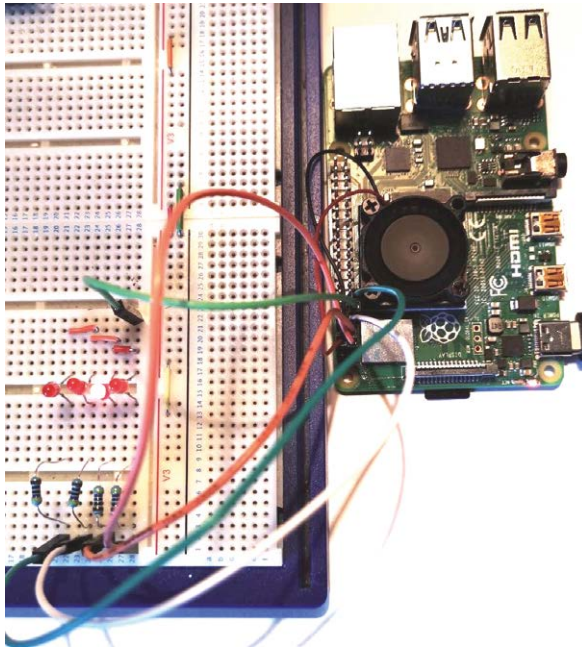


Figure 5.40 Constructing the project on a breadboard

Just a reminder, if you are using a Gmail account, you will have to enable **less secure access** via your Google account settings as described in earlier projects using the email nodes.

5.10 Project 20 – Traffic lights simulator

Description: In this project, we will simulate a set of traffic lights. 3 LEDs are connected to the GPIO ports of the Raspberry Pi: a Red LED, an Orange LED, and a Green LED. The LEDs are turned ON and OFF as in the following PDL sequence:

```
DO FOREVER
  Turn Red LED ON for 17 seconds
  Turn Red and Orange LED ON for 3 seconds
```

```
Turn Green LED ON for 15 seconds
Turn Orange LED ON for 3 seconds
ENDDO
```

Aim: The aim of this project is to show how a simple traffic light simulator program can be developed with Node-RED using 3 LEDs connected to a Raspberry Pi.

Block diagram: Figure 5.41 shows the block diagram of the project.

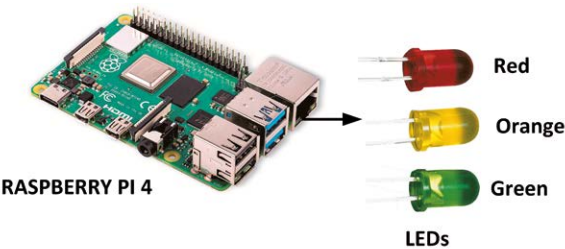


Figure 5.41 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 5.42. The LEDs are connected to the following Raspberry Pi port pins through 470 Ohm current limiting resistors:

LED	Raspberry Pi Port Name	Raspberry Pi Port Pin Number
Red	GPIO 2	3
Orange	GPIO 3	5
Green	GPIO 4	7

Port pin number 9 of the Raspberry Pi is the ground pin that is connected to the cathode pins of all the LEDs.

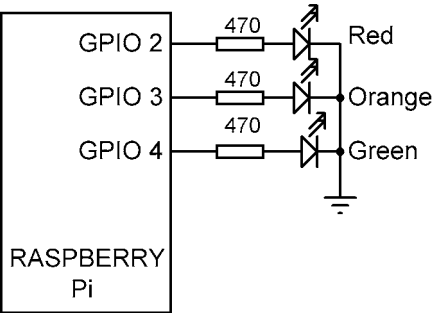


Figure 5.42 Circuit diagram of the project

Node-RED flow program: Figure 5.43 shows the flow program. In this program 11 nodes are used: an **inject** node to start the simulation, 4 **delay** nodes to generate the required timing delays, 3 **trigger** nodes, and 3 **rpi gpio out** nodes.

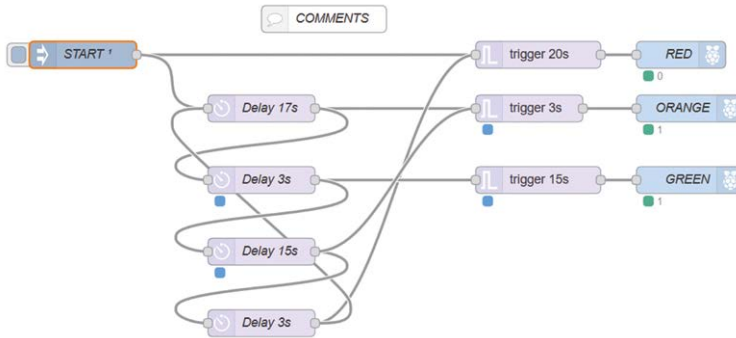


Figure 5.43 Flow program of the project

The steps are as follows:

- Create an **inject** node and name it as **START**
- Create 4 **delay** nodes as shown in Figure 5.43. These nodes will generate the required delay timings. For example, when the Red LED is triggered to stay ON for 20 seconds. After a delay of 17 seconds, the Orange LED is also turned ON so that both the Red and Orange are ON for 3 seconds. After this time the Green LED is triggered to stay ON for 15 seconds, and after this time the Orange LED is turned ON for 3 seconds, and then back to the Red LED. This sequence is repeated forever. Figure 5.44 shows a typical **delay** node configuration.

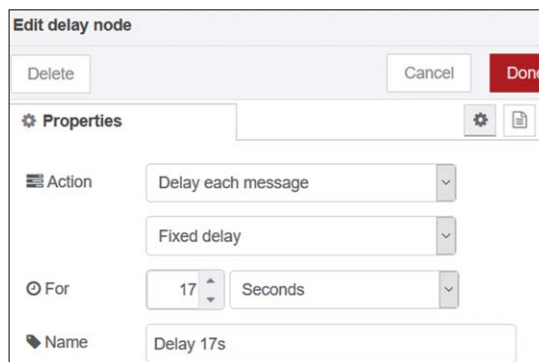


Figure 5.44 17 seconds delay node

- Create 3 **trigger** nodes that send out 1, wait for the specified time, and then send out 0. Figure 5.45 shows as an example of how the 20s **trigger** node is configured.

The screenshot shows the 'Properties' panel for a 'trigger' node in Node-RED. The configuration is as follows:

- Send:** A dropdown menu showing 'a' and 'z' with the value '1'.
- then:** A dropdown menu showing 'wait for'.
- Delay:** A text input field containing '20' and a unit dropdown menu set to 'Seconds'.
- extend delay if new message arrives:** An unchecked checkbox.
- then send:** A dropdown menu showing 'a' and 'z' with the value '0'.
- Reset the trigger if:** A list of conditions:
 - msg.reset is set
 - msg.payload equals optional (with a text input field containing 'optional')
- Handling:** A dropdown menu set to 'all messages'.

Figure 5.45 Configuring a trigger node

- Create 3 **rpi gpio out** nodes for the GPIO pins GPIO 2, GPIO 3, and GPIO 4, and name them as **RED**, **ORANGE**, and **GREEN** respectively. As an example, Figure 5.46 shows the configuration of node GREEN.

The screenshot shows the 'Properties' panel for an 'rpi gpio out' node in Node-RED, configured for node GREEN. The configuration is as follows:

- Pin Selection:** A table of GPIO pins with radio buttons. The selected pin is GPIO04 - 7.

SCL1 - GPIO03 - 5	6 - Ground
GPIO04 - 7	8 - GPIO14 - TxD
Ground - 9	10 - GPIO15 - RxT
GPIO17 - 11	12 - GPIO18
GPIO27 - 13	14 - Ground
GPIO22 - 15	16 - GPIO23
3.3V Power - 17	18 - GPIO24
MOSI - GPIO10 - 19	20 - Ground
MISO - GPIO09 - 21	22 - GPIO25
SCLK - GPIO11 - 23	24 - GPIO8 - CE0
Ground - 25	26 - GPIO7 - CE1
SD - 27	28 - SC
GPIO05 - 29	30 - Ground
GPIO06 - 31	32 - GPIO12
GPIO13 - 33	34 - Ground
GPIO19 - 35	36 - GPIO16
GPIO26 - 37	38 - GPIO20
Ground - 39	40 - GPIO21
- Type:** A dropdown menu set to 'Digital output'.
- Initialise pin state?:** A checked checkbox.
- Initial level of pin - low (0):** A dropdown menu set to 'low (0)'.

Figure 5.46 Configuration of node GREEN

- Join all the nodes as shown in Figure 5.43, and click **Deploy**. Click the button of node **inject** to start the simulation. You should see the 3 LEDs simulating the traffic lights with the timing as specified at the beginning of this project.

Notice that in this project the **comment** node is also introduced. This node helps us to write comments about a project. In this example project, the contents of the **comment** node are shown in Figure 5.47.

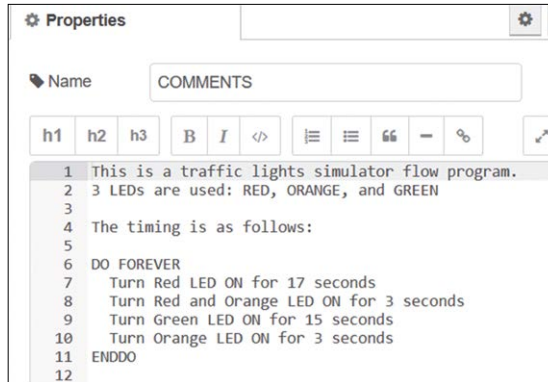


Figure 5.47 Contents of the comment node

This project was constructed on a breadboard as in the previous projects.

5.11 Project 21 – Push-Button switch input

Description: This is a very simple project where a push-button switch (button for short) and an LED are connected to the Raspberry Pi. The project turns ON the LED when the button is pressed.

Aim: The aim of this project is to show how a button can be used in a Node-RED based project.

Background information: Buttons are commonly used in microcontroller based projects in order to change the state of an input port pin. The processor detects when the button is pressed and then executes the required code.

Buttons can be used in two modes: Normally LOW, and normally HIGH. In a normally LOW mode, the output state of the button is at logic 0 by default and goes to logic 1 when the button is pressed. The output stays at this state until the button is released. The circuit diagram in Figure 5.48 shows a normally LOW button.

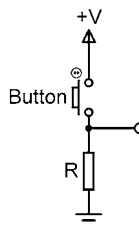


Figure 5.48 Normally LOW button

In a normally HIGH button, the state of the button is at logic 1 by default and goes to logic 0 when the button is pressed. The output stays at this state until the button is released. The circuit diagram in Figure 5.49 shows a normally HIGH button.

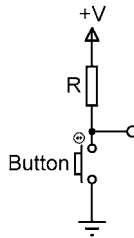


Figure 5.49 Normally HIGH button

The button used in this project is a small component having 4 pins as shown in Figure 5.50. Actually, the button is a 2 pin device where the pins are paralleled for convenience.



Figure 5.50 Small button and its connection diagram

Mechanical switches have bouncing problems. This happens because when a switch is toggled, its mechanical contacts bounce (move from one state to another state) until they settle. Although this settling time may seem to be very small (around 10ms), it is actually a very long time when the processing time of a microcontroller is considered. Special switch debouncing circuits can be used to eliminate this bouncing effect. Luckily, the **rpi gpio in** node provides a parameter to set the switch debouncing time so that the switch contacts are not read until this time expires. This makes sure that the correct switch output state is read by the microcontroller.

Block diagram: Figure 5.51 shows the block diagram of the project.



Figure 5.51 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 5.52 where the LED is connected to port pin GPIO 2. Port pin GPIO 3 is connected to one side of the button, while the other side of the button is connected to the ground pin of Raspberry Pi.

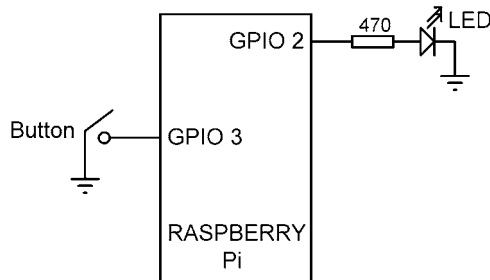


Figure 5.52 Circuit diagram of the project

Node-RED flow program: Figure 5.53 shows the flow program. In this program 2 nodes are used:

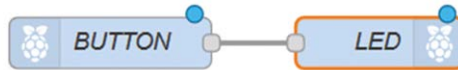


Figure 5.53 Flow program of the project

The steps are:

- Create a **rpi gpio in** node and name it as **BUTTON**, set the pin name to GPIO 3, and set the resistor to pulldown.
- Create a **rpi gpio out** node and name it as **LED**, set the pin name to GPIO 3, and initialize the pin level to 0
- Join the two nodes as shown in Figure 5.51, and click **Deploy**.

You should see that the LED is ON, and pressing the button turns OFF the LED. Notice that in this flow program we want the opposite. i.e. the LED should be OFF normally, and pressing the button should turn it ON. A modified flow program that will work correctly is shown in Figure 5.54

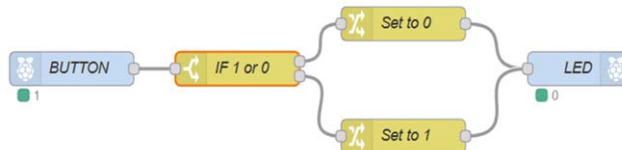


Figure 5.54 Modified flow program

This modification can be made as explained in the following steps:

- Create a **rpi gpio in** node as above and name it as **BUTTON**, set the pin name to GPIO 3, and set the resistor to pulldown
- Create a **switch** node and name it as **IF 1 or 0**. Configure this node such that if the input is 1, direct it to the top output, otherwise, direct it to the second output. To direct to the second output, click add, and select **otherwise** as shown in Figure 5.54. Click **Done**.



Figure 5.55 Configure the switch node

- Create a **change** node and name it as **Set to 0**. Configure this node to return 1 as shown in Figure 5.56. Click **Done**



Figure 5.56 Configure the change node

- Create another **change** node and name it as **Set to 1** and set the output to 1. Click **Done**
- Create a **rpi gpio out** node and name it as **LED**, set the pin name to GPIO 3, and initialize the pin level to 0
- Join the 5 nodes as shown in Figure 5.54, and click **Deploy**.

You should now see that the LED is OFF, and pressing the button turns the LED ON.

Figure 5.57 shows the circuit constructed on a breadboard.

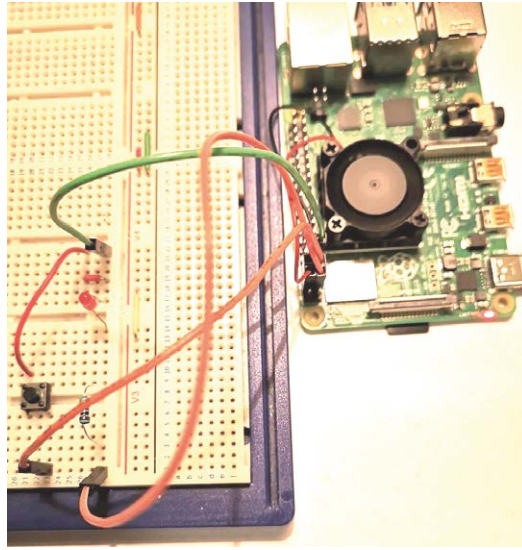


Figure 5.57 Circuit constructed on a breadboard

5.12 Project 22 – Changing LED brightness – The PWM output

Description: This is a very simple project where the brightness of an LED is altered by changing the average voltage and hence average current through the LED.

Aim: The aim of this project is to show how a PWM waveform can be generated using Node-RED.

Background information: PWM (Pulse Width Modulation) is a technique used commonly in electronic circuits in order to change the average voltage supplied to a load. For example, the average voltage supplied to a heater can be changed by sending it a fixed amplitude PWM voltage waveform and then changing the characteristics of this waveform.

A PWM waveform is basically a positive square waveform where the ON time (also called the MARK) and the OFF time (also called the SPACE) can be changed. Figure 5.58 shows a typical PWM waveform with a period of the T. The ratio of the ON time to the period is known as the Duty Cycle of the waveform and by changing the duty cycle we can effectively change the average voltage supplied to the load.

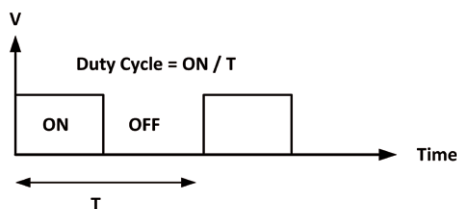


Figure 5.58 Typical PWM waveform

The duty cycle is commonly represented as a percentage. For example, a 50% duty cycle corresponds to the situation where the ON time is equal to the OFF time. In a 100% duty cycle, the waveform is always ON (i.e. there is no OFF time). Similarly, in a 25% duty cycle, the duration of the ON time is a quarter of the period of the waveform. Figure 5.59 shows PWM waveforms with different duty cycles.

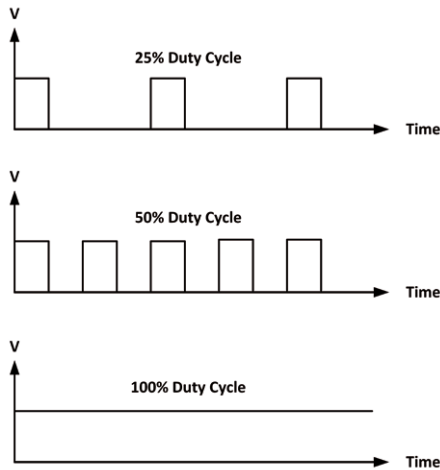


Figure 5.59 PWM waveform with different duty cycles

Circuit diagram: The circuit diagram of the project is as in Figure 5.4 where an LED is connected to port pin GPIO 2 of Raspberry Pi.

Node-RED flow program: Figure 5.60 shows the flow program. In this program, 6 nodes are used: 5 **inject** nodes with different duty cycle settings and a **rpi gpio out** node.

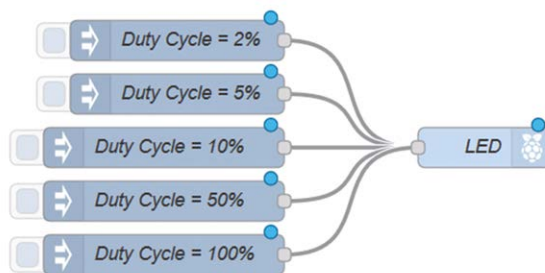


Figure 5.60 Flow program of the project

The steps are:

- Create 5 **inject** nodes, set the payloads to numeric values of 2, 5, 10, 50, 100. These will be the duty cycles of the PWM waveform. Name each inject node with their corresponding duty cycles as shown in Figure 5.60. For example, the configuration of the node with a 50% duty cycle is shown in Figure 5.61

Payload: 50
 Topic:
☐ Inject once after 0.1 seconds, then
 Repeat: none
 Name: Duty Cycle = 50%

Figure 5.61 Configuring for 50% duty cycle

- Create a rpi gpio out node and name it as LED, set the port name to GPIO 2, and set the Type to PWM output as shown in Figure 5.61. Click Done

3.3V Power - 1	2 - 5V Power
SDA1 - GPIO02 - 3	4 - 5V Power
SCL1 - GPIO03 - 5	6 - Ground
GPIO04 - 7	8 - GPIO14 - TxD
Ground - 9	10 - GPIO15 - RxD
GPIO17 - 11	12 - GPIO18
GPIO27 - 13	14 - Ground
GPIO22 - 15	16 - GPIO23
3.3V Power - 17	18 - GPIO24
MOSI - GPIO10 - 19	20 - Ground
MISO - GPIO09 - 21	22 - GPIO25
SCLK - GPIO11 - 23	24 - GPIO8 - CE0
Ground - 25	26 - GPIO7 - CE1
SD - 27	28 - SC
GPIO05 - 29	30 - Ground
GPIO06 - 31	32 - GPIO12
GPIO13 - 33	34 - Ground
GPIO19 - 35	36 - GPIO16
GPIO26 - 37	38 - GPIO20
Ground - 39	40 - GPIO21

PWM output

Figure 5.62 Configure the rpi gpio out node

Join the nodes as in Figure 5.60. Click inject node 2% and you should see that the LED is very dim. Clicking higher duty cycles should make the LED brighter.

5.13 Summary

In this chapter, we have learned how to connect external components to Raspberry Pi GPIO ports and then how to access these components from Node-RED programs.

In the next chapter, we will be looking at how to connect an LCD to our Raspberry Pi and we will also develop further projects using an LCD.

Chapter 6 • Using LCDs with Node-RED

6.1 Overview

In microcontroller systems, the output of a measured variable is usually displayed using LEDs, 7-segment displays, or LCD type displays. LCDs have the advantages that they can be used to display alphanumeric or graphical data. Some LCDs have 40 or more character lengths with the capability to display several lines. Some other LCD displays can be used to display graphics images. Some modules offer colour displays while some others incorporate backlighting so that they can be viewed in dimly lit conditions.

There are basically two types of LCDs as far as the interface technique is concerned: parallel LCDs and serial LCDs. Parallel LCDs (e.g. Hitachi HD44780) are connected to a microcontroller using more than one data line and the data is transferred in parallel form. It is common to use either 4 or 8 data lines. Using a 4 wire connection saves I/O pins but it is slower since the data is transferred in two stages. One of the most commonly used parallel LCD standard is the HD44780.

The programming of a parallel LCD is usually a complex task and requires a good understanding of the internal operation of the LCD controllers, including the timing diagrams. Fortunately, most high-level languages provide special library commands for displaying data on alphanumeric as well as on graphical LCDs. All the user has to do is connect the LCD to the microcontroller, define the LCD connection in the software, and then send special commands to display data on the LCD.

Serial LCDs are connected to the microcontroller using only a few data lines. Serial LCDs are much easier to use but they cost more than the parallel ones. One of the most commonly used serial LCDs makes use of the I²C bus protocol where only two wires are required for communication, namely SDA (data) and SCL (clock). There is a master device on the bus and there can be a number of slave devices where each device has a unique address so that it can be identified. Figure 6.1 shows a typical I²C bus with a master and two slave devices. Notice that the bus must be terminated with pull-up resistors to the power supply. Interested readers should be able to find many articles, application notes, and data sheets on the Internet on I²C. In this chapter, we will be using an I²C based LCD in Node-RED based Raspberry Pi projects.

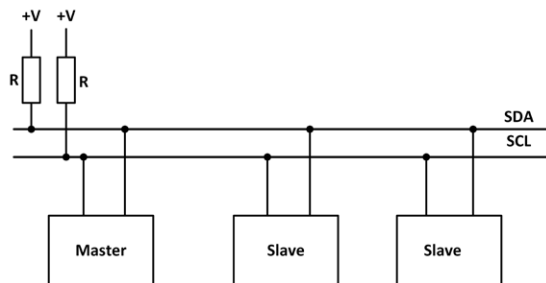


Figure 6.1 I²C Bus structure with a master and 2 slave devices

6.2 Raspberry Pi I²C ports

Raspberry Pi provides two I²C ports SCL0 and SCL1 at the following GPIO pins:

I ² C Name	GPIO Port Pin	Signal
SCL0	GPIO 0	SDA0
SCL0	GPIO 1	SCL0
SCL1	GPIO 2	SDA1
SCL1	GPIO 3	SDA2

It is recommended to use SCL1 in I²C based applications. The two I²C lines are internally pulled-up to +3.3V power supply using 1.8K resistors.

The I²C on your Raspberry Pi 4 must be enabled using the raspi-config utility before it can be used. The steps are:

- Start raspi-config on your Raspberry Pi 4 by entering the following command

```
pi@raspberrypi:~ $ sudo raspi-config
```

- Move down to item **5 Interfacing Options**
- Move down to **P5 I²C** (see Figure 6.2) and press Enter
- Select **Yes** to enable the I²C interface, and click **OK** and exit from raspi-config utility

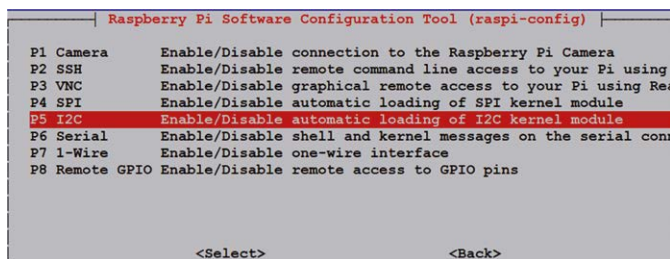


Figure 6.2 Move to P5 I²C

Some of the I²C devices operate with +5V. It is important to be careful when connecting such devices to your Raspberry Pi since the voltage at the I/O pins of the Raspberry Pi must not exceed +3.3V, otherwise, the Raspberry Pi can be damaged. It is safe to connect an I²C device that operates with +5V to your Raspberry Pi if the device does not have any internal pull-up resistors on its I²C lines. If the I²C lines of the device are connected internally to +5V, then you must not connect such a device to your Raspberry Pi without using voltage level converters circuits to lower the voltage from +5V to +3.3V.

6.3 I²C LCD

The commonly available I²C LCD consists of a standard HD44780 compatible 16x2 character display that is operated in 4-bit mode. A small board is plugged-in at the back of the LCD board. This board has a PCF8574 chip that converts I²C serial data to parallel form for the LCD display. Although the slave address of the LCD can be changed from 0x20 to 0x27, its default value is 0x27. The LCD module operates with +5V and it has no internal pull-up resistors on its I²C lines. The module can therefore directly be connected to the Raspberry Pi I²C lines.

To test the LCD module and find its slave address follow the steps given below:

- Connect the SDA and SCL pins of the LCD modules to GPIO 2 and GPIO 3 port pins of the Raspberry Pi respectively
- Connect the Vcc and GND pins of the LCD module to +5V and GND pins of the Raspberry Pi respectively
- Enter the following command on your Raspberry Pi:

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
```

- You should get a display similar to the one shown in Figure 6.3. Notice in this figure that the slave address of your LCD module is 0x27

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  27  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

Figure 6.3 Address of the LCD module is 0x27

- If the address of the LCD module is not displayed it means that there are some errors. You should check your connections and try again. You may also have to install the I²C tools by entering the following commands:

```
pi@raspberrypi:~ $ sudo apt-get install python-smbus
pi@raspberrypi:~ $ sudo apt-get install i2c-tools
```

6.4 Installing the I²C LCD software on Node-RED

There are several I²C LCD devices. The one we will be using in this project is HD44780 compatible 16 characters by 2 rows display, which has a small board at its back with a PCF8574 chip (see Figure 6.4).

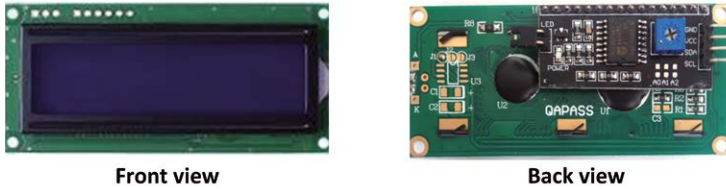


Figure 6.4 I²C LCD used in the project

A compatible I2C LCD node must be installed so that it is available in the Node-RED palette. The steps to install it are as follows:

- Start Node-RED and click **Menu -> Manage palette** and then click **Install**
- Enter **node-red-contrib-lcd20x4-i2c** in the search path and click **install**
- You should now see a new node called **LCD20x4 I2C**

The LCD node can be used for LCDs having up to 4 lines, and it accepts an object **msg.payload.msgs**. If the object passed does not contain 4 lines the difference is filled with blank lines. The following points are important while using this LCD node:

- **msg** must be a string no more than 20 characters long. Scrolling will be done if more than 20 characters are sent to the LCD.
- A **position** object can be specified between 1 and 20 (default is 1 if not specified) which is used for offsetting the text (spaces can be inserted instead of using this object)
- A **center** object is optional and it must be specified as a Boolean value, passed as a string

If there is an error in the data sent to the LCD then an error message will be displayed on the LCD screen.

An example is given below to show how the LCD node can be used.

Example

Develop a flow program to display the text **Hello From** at the top row and **My LCD** in the second row.

Solution

The required flow program is shown in Figure 6.5.



Figure 6.5 Flow program of the example

The steps are as follows:

- Create an **inject** node and name it as **Click**
- Create a **function** node and name it as **Display**. Enter the following statements inside this node (see Figure 6.6).

```
msg.payload={msgs:[
    {msg:"Hello From"},
    {msg:"My LCD"}
]};
return msg;
```

The message in the first part of the statement enclosed with {} is sent to the top row, and the second message enclosed with {} is sent to the bottom row of the LCD.



Figure 6.6 Contents of function node

- Click, drag and drop the **LCD20x4 I2C** node to the workspace and change its name to **LCD**. Set the I2C address to 0x27 and click **Done**
- Join the nodes as shown in Figure 6.5, and click **Deploy**. You should see the text displayed as shown in Figure 6.7

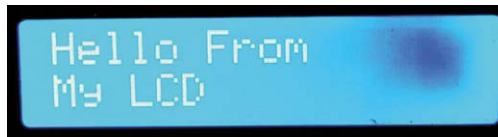


Figure 6.7 Text displayed on the LCD

The LCD object format for a two-row LCD (16 x 2) is as follows (see link: <https://flows.nodered.org/node/node-red-contrib-lcd20x4-i2c>):

```
msg.payload = {
  msgs: [
    {
      msg: "string",
```

```

        pos: number,
        center: "boolean"
    },
    {
        msg: "string",
        pos: number,
        center: "boolean"
    },
]
};

```

The data to be displayed is entered after the keyword **msg**. The first message is for the top row, and the second message is for the bottom row of the LCD.

6.5 Project 23 – Displaying current time on the LCD

Description: In this project, the current time is displayed on the LCD every second in the following format:

```

Current Time
Time=hh:mm:ss

```

Aim: The aim of this project is to show how the current time can be extracted and then displayed on the LCD every second.

Circuit diagram: Figure 6.8 shows the circuit diagram of the project. The LCD is directly connected to the I²C pins of the Raspberry Pi.

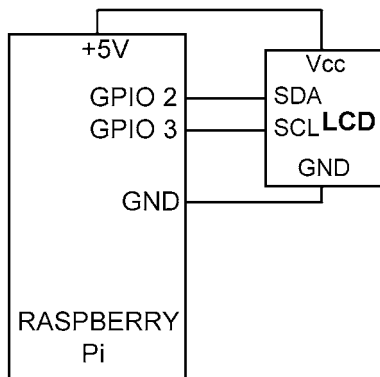


Figure 6.8 Circuit diagram of the project

Node-RED flow program: Figure 6.9 shows the flow program which consists of 3 nodes: an **inject** node, a **function** node, and an **LCD20x4 I2C** node.



Figure 6.9 Flow diagram of the project

The steps are as follows:

- Create an **inject** node named **Click**, and set its interval to one second
- Create a **function** node named **Display**, and enter the following commands (see Figure 6.10), click **Done**. Variable **LocalTime** extracts the current local time. Variable **Tim** constructs the string Time=hh:mm:ss. The first row of the LCD displays the text **Current Time**, and the second row displays the actual time:

```

var LocalTime = new Date();
var Tim = "Time=" + LocalTime.toLocaleTimeString();
msg.payload={msgs:[
    {msg:"Current Time"},
    {msg:Tim, pos:1}
]};

return msg;

```



Figure 6.10 Configure the function node

- Create an **LCD20x4 I2C** node named **LCD** and set the I2C Address to 0x27
- Join all 3 nodes as shown in Figure 6.9, click **Deploy**. Click the button of the **inject** node. You should see the time displayed on the LCD every second as shown in Figure 6.11



Figure 6.11 Displaying the time every second

6.6 Project 24 – Displaying local temperature and humidity on the LCD

Description: In this project, the current weather forecast is received and the local temperature and humidity are displayed on the LCD every 10 seconds.

Aim: The aim of this project is to show how the local temperature and humidity can be displayed on the LCD.

Circuit diagram: The circuit diagram of the project is the same as in Figure 6.8.

Node-RED flow program: Figure 6.12 shows the flow program which consists of 4 nodes: an **inject** node to request reading every 10 seconds, an **openweathermap** node to get the local weather map, a **function** node to extract the local temperature and humidity, and an **LCD20x4 I2C** node to display the temperature and humidity.



Figure 6.12 Flow diagram of the project

The steps are as follows:

- Create an **inject** node with an interval of 10 seconds and name it as **Click**
- Create an **openweathermap** node and name it as **Temp & Hum**. Set the API key and set your local city and country names. Click **Done**
- Create an **LCD20x4 I2C** node and name it as **LCD**. Set the I2C address to 0x27 as before and click **Done**
- Join the 4 nodes and click **Deploy**.
- Click the button of the **inject** node. You should see the local temperature and humidity displayed on the LCD as shown in Figure 6.13.



Figure 6.13 Displaying the local temperature and humidity

6.7 Project 25 – Displaying dice numbers

Description: In this project, two random numbers are generated between 1 and 6 and are displayed on the LCD every time the button of the inject node is clicked. The numbers are displayed for 5 seconds. After this time the display is cleared and message **Ready** is displayed. At this point, the user can click the button of the inject node to generate two new

random numbers. This project can be used in games that are played with dice.

Aim: The aim of this project is to show how two random numbers can be generated and then combined and displayed on the LCD.

Circuit diagram: The circuit diagram of the project is the same as in Figure 6.8.

Node-RED flow program: Figure 6.14 shows the flow program which consists of 8 nodes: an **inject** node to start generating two random numbers, 2 **random** nodes to generate two random numbers between 1 and 6, a **join** node to join the two generated numbers, a **function** node to format the numbers for displaying on the LCD, a **delay** node to generate 5 seconds of delay so that the message Ready can be displayed after 5 seconds, and a **function** node which actually displays the message **Ready**.

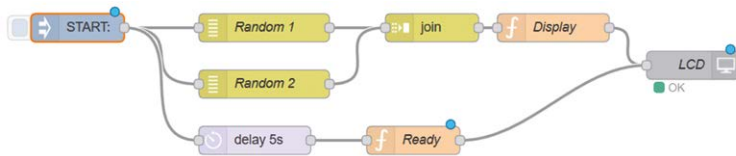


Figure 6.14 Flow program of the project

The steps are as follows:

- Create an **inject** node and name it as **START**
- Create two **random** nodes and name them as **Random 1** and **Random 2**. Set the nodes to generate random numbers between 1 and 6, click **Done**
- Create a **join** node and configure it as shown in Figure 6.15. This node will join the two random numbers into an array. Click **Done**.

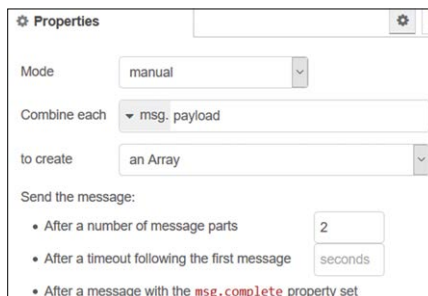


Figure 6.15 Configure the join node

- Create a **function** node and name it as **Display**. Enter the following statements into this node (see Figure 6.16). These statements extract the two numbers from the array and combine them into a single variable called **disp**. The contents of **disp** are then displayed on the first row of the LCD. Click **Done**.

```
var first = msg.payload[0];
var second = msg.payload[1];
var disp = first + " " + second;
msg.payload={msgs:[
    {msg:disp}
]};
return msg;
```

- Create an **LCD20x4 I2C** node and name it as **LCD**. Set the I2C address to 0x27 as before and click **Done**
- Create a 5-second **delay** node
- Create a **function** node and name it as **Ready**. Enter the following statements into this node. This node will display the message **Ready** on the LCD:

```
msg.payload={msgs:[
    {msg:"Ready"}
]};
return msg;
```

- Join all the nodes as shown in Figure 6.14 and click **Deploy**.
- Click the button of the **inject** node. You should see two numbers generated between 1 and 6. After 5 seconds the display will clear and message **Ready** will be displayed so that clicking the button again will generate a new set of numbers. Figure 6.16 shows an example output on the LCD.



Figure 6.16 Example output on the LCD

6.8 Project 26 – Event counter with LCD

Description: This is an event counter project. Events are assumed to occur when port pin GPIO 15 of Raspberry Pi goes from logic 1 to logic 0. In the project, events are simulated by connecting a push-button switch to GPIO 15 and by pressing the button. The program counts the events occurring and displays the total at any time on the LCD.

Aim: The aim of this project is to show how a simple event counter program can be developed using Node-RED.

Circuit diagram: The two-button pins are connected to GPIO 15 and to ground respectively.

Node-RED flow program: Figure 6.17 shows the flow program which consists of 3 nodes: a **rpi gpio in** node to receive the button inputs, a **function** node to update the counts, and an **LCD20x4 I2C** node to display the count.



Figure 6.17 Flow program of the project

The steps are as follows:

- Create a **rpi gpio in** node and name it as **BUTTON**. Set the pin number GPIO 15, set **Resistor** to pullup and **Debounce** to 25ms. Notice that **Debounce** is set to 25ms to eliminate the contact bouncing problems associated with the mechanical switches as described in Chapter 5. Click to read the initial state of the pin. Click **Done**
- Create a **function** node and name it as **Display**. Enter the following statements inside this node (see Figure 6.18). Notice that we are using a context variable here. The statement:

```
var count=context.get('count') || 0;
```

means that if **count** does not exist in the context object, then make variable **count** zero, otherwise assign the stored value to **count**. The value of **count** is incremented every time the button is pressed (i.e. msg.payload goes to 0). **Count** is then converted into string and sent to the LCD:

```
var count=context.get('count') || 0;
if(msg.payload == 0)
{
    count++;
    context.set('count',count);
    var cnt = "Events = " + count.toString();
    msg.payload={msgs:[
        {msg:cnt}
    ]};
    return msg;
}
else
    return null;
```



```
var count=context.get('count') || 0;

if(msg.payload == 0)
{
    count++;
    context.set('count',count);
    var cnt = "Events = " + count.toString();
    msg.payload={msg:cnt}
    return msg;
}
else
    return null;
```

Figure 6.18 Configure function node

- Create an **LCD20x4 I2C** node and name it as **LCD**. Set the I2C address to 0x27 and click **Done**
- Join all the nodes as in Figure 6.17 and click **Deploy**. Every time the button is pressed, the event count will be incremented by one and the total value will be displayed on the screen as shown in Figure 6.19.

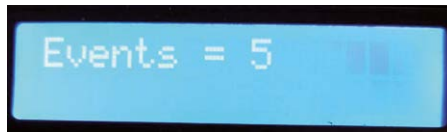


Figure 6.19 Displaying the total events at any time

6.9 Project 27 – DHT11 temperature and humidity sensor with LCD Display

Description: In this project, we use the popular DHT11 digital sensor to measure the ambient temperature and humidity and display them on the LCD every 5 seconds.

Aim: The aim of this project is to show how the DHT11 temperature and humidity sensor can be used in a Node-RED project.

Block diagram: Figure 6.20 shows the block diagram of the project.



Figure 6.20 Block diagram of the project

Background information: DHT 11 (or DHT22) is one of the popular and most commonly used digital temperature and humidity sensors. The sensor basically has 3-pins as shown in Figure 6.21. Although some sensor versions have 4 pins, one of the pins is not used. The

DHT11 is made up of two parts: a capacitive humidity sensor, and a thermistor temperature sensor. Because the output is digital, the chip also includes an ADC to convert the analog signals into digital.

The basic specifications of DHT11 are:

- 3V to 5V power supply
- Low cost
- 2.5mA max current (during conversion)
- 20 - 80% humidity range with 5% accuracy
- 0 - 50°C temperature range with $\pm 2^{\circ}\text{C}$ accuracy
- 1 Hz sampling rate (once every second)

DHT22 is similar to DHT11, it has the same size and shape, but it offers wider measurement ranges and is also more accurate than the DHT11. In this project, we will be using a DHT11.

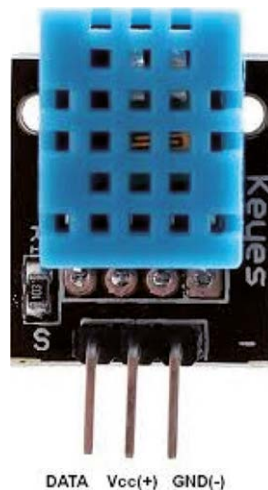


Figure 6.21 The DHT11 sensor

Installing the DHTxx Node-RED node: The steps to install the DHTxx (i.e. DHT11 and DHT22) Node-RED node are as follows:

- Click **Menu -> Manage** palette and click **Install**
- Enter **node-red-contrib-dht-sensor** and click **install** (see Figure 6.22)
- You should see a new node called **rpi-dht22** in the Node Palette

We are now ready to develop our flow-program to read and display the ambient temperature and humidity.

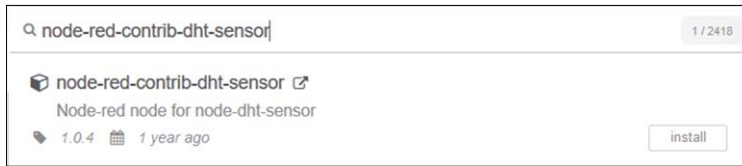


Figure 6.22 Install the DHTxx node

Circuit diagram: The circuit diagram of the project is shown in Figure 6.23. The I²C LCD is connected to Raspberry Pi as in the earlier projects in this Chapter. The DHT11 is connected to the Raspberry Pi as follows:

DHT11 Pin	Raspberry Pi Pin
Data (S)	GPIO 15
Vcc	+3.3V
Gnd (-)	GND

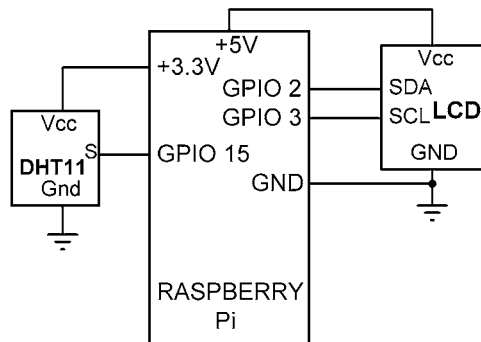


Figure 6.23 Circuit diagram of the project

Node-RED flow program: Figure 6.24 shows the flow program which consists of 3 nodes: a **rpi gpio in** node to receive the button inputs, a **function** node to update the counts, and an **LCD20x4 I2C** node to display the count.

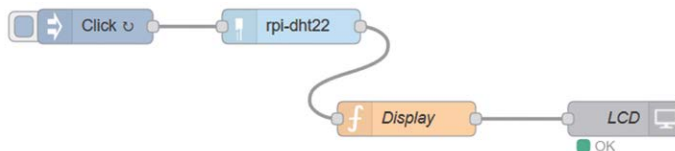


Figure 6.24 Flow program of the project

The steps are as follows:

- Create an **inject** node with the name **Click** and set the interval to 5 seconds

- Create a **rpi-dht22** node and configure it as shown in Figure 6.25. Set the **Sensor model** to DHT11, **Pin numbering** to BCM GPIO, and the pin number to 15 (GPIO 15), click Done



Figure 6.25 Configure the *rpi-dht22* node

- Create a **function** node and name it as **Display**. Enter the following statements inside the function (see Figure 6.26). Variables **T** and **H** receive the temperature and humidity readings respectively from the DHT11 every 5 seconds. Temperature is displayed at the top row, while the humidity is displayed at the bottom row of the LCD:

```
T = msg.payload;
H = msg.humidity;
Temp = "T = " + T + "C";
Hum = "H = " + H + "%";

msg.payload={msg:[
    {msg: Temp},
    {msg: Hum}
]};

return msg;
```

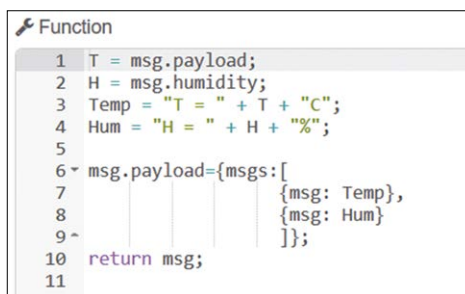


Figure 6.26 Configure the function node

- Create the LCD node as before, join all the nodes and click **Deploy**.
- Click the button of the **inject** node. You should see the temperature and humidity updated and displayed every 5 seconds (see Figure 6.27)

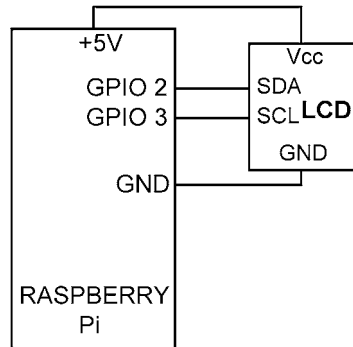


Figure 6.27 Displaying the temperature and humidity

Notice that if you connect a **debug** node to the output of node `rpi-dht22`, you will only see the temperature displayed in the payload. If, however, you set the Output of the debug node to **complete msg object**, you will see both the temperature and the humidity displayed as shown in Figure 6.28

```

12/12/2019, 09:44:08 node: a69647b5.0037e8
rpi-dht22 : msg : Object
{ topic: "rpi-dht22", payload:
  "15.00", _msgid: "23c59261.47d7ce",
  humidity: "23.00", isValid: true ... }
  
```

Figure 6.28 Debug node output

Figure 6.29 shows the circuit built on a breadboard and connected to the Raspberry Pi using jumper wires.

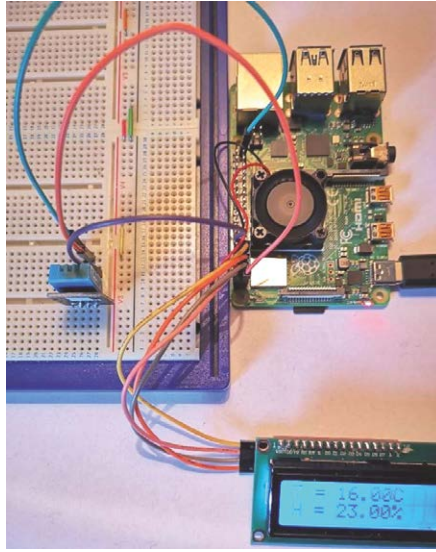


Figure 6.29 Circuit built on a breadboard

6.10 Project 28 – Ultrasonic distance sensor with LCD display

Description: In this project, an ultrasonic distance sensor module is used and the distance to any obstacle in front of the sensor is displayed on the LCD.

Aim: The aim of this project is to show how an ultrasonic sensor can be used in a Node-RED project.

Background information: Ultrasonic distance sensor modules are used applications where it is required to measure the distance to an object (e.g. an obstacle) in-front of the sensor module. The most commonly used ultrasonic sensor is the HC-SR04 (see Figure 6.30). This is a 4-pin module with the following pins:

Echo: echo pin

Trig: trigger pin

Vcc: Power supply pin (+5V)

GND: power supply ground pin



Figure 6.30 The HC-SR04 ultrasonic distance sensor module

The measurement range of the HC-SR04 sensor is 2cm to 400cm. The basic principle of operation of the ultrasonic sensor module is as follows (see Figure 6.31):

- A 10µs trigger pulse is sent to the module
- The module then sends eight 40kHz square wave signals to the target and sets the echo pin HIGH
- The program starts a timer
- The signal hits the target and echoes back to the module
- When the signal is returned to the module the echo pin goes LOW
- The timer is stopped
- The duration of the echo signal is calculated and this is proportional to the distance to the target

The distance to the object is calculated as follows:

$$\text{Distance to object (in metres)} = (\text{duration of echo time in seconds} * \text{speed of sound}) / 2$$

The speed of sound is 340 m/s, or 0.034 cm/µs

Therefore,

$$\text{Distance to object (in cm)} = (\text{duration of echo time in } \mu\text{s}) * 0.034 / 2$$

or,

$$\text{Distance to object (in cm)} = (\text{duration of echo time in } \mu\text{s}) * 0.017$$

For example, if the duration of the echo signal is 294 microseconds then the distance to the object is calculated as follows:

$$\text{Distance to object (cm)} = 294 * 0.017 = 5\text{cm}$$

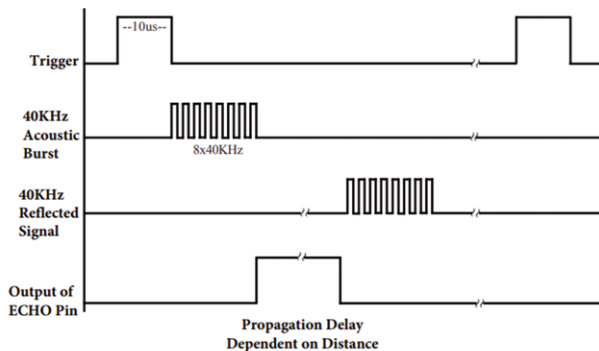


Figure 6.31 Operation of the ultrasonic sensor module

Installing the ultrasonic sensor Node-RED node: The steps to install the HC-SR04 sensor Node-RED node are as follows:

- Click **Menu -> Manage** palette and click **Install**
- Enter **node-red-node-pisrf** and click **install**
- You should see a new node called **rpi srf** in the Node Palette
- We are now ready to develop our flow-program to read and display the distance

Block diagram: The block diagram of the project is shown in Figure 6.32.



Figure 6.32 Block diagram of the project

Circuit diagram: Figure 6.33 shows the circuit diagram. Raspberry Pi port pins GPIO 15 and GPIO 18 are connected to the Trig and Echo pins of the sensor respectively. Notice that the sensor operates with +5V and its Echo output voltage level goes up to +5V. This is too high for the Raspberry Pi inputs and as a result, a resistive potential divider circuit is used to lower the sensor output voltage to +3.3V. The VCC and GND pins of the sensor are connected to +5V and GND pins of Raspberry Pi respectively.

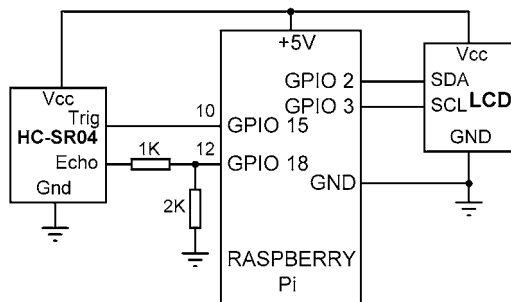


Figure 6.33 Circuit diagram of the project

Node-RED flow program: Figure 6.34 shows the flow program which consists of 3 nodes: a **rpi srf** node to read the distance from the sensor, a **function** node to format the data to be displayed, and an **LCD** node.



Figure 6.34 Flow program of the project

The steps are as follows:

- Click, drag and drop the **rpi srf** node into the workspace. Configure it as shown in Figure 6.35. Note that by default the distance is measured every 0.5 seconds, but this can be changed if desired. Set the pin numbers to where the Trig and Echo pins are connected on the Raspberry Pi. It is important to be careful here since the pin numbers required here are the physical pin numbers of the header and not the GPIO pin names. In this project, Trig and Echo pins are connected to GPIO pins 15 and 18 respectively, which correspond to physical pin numbers 10 and 12 respectively.

Properties

Pins 10,12

Repeat (S) 0.5

Topic SRF

Name Name

Tip: Pins MUST be a comma separated list of the 2 GPIO connector pin numbers that are connected to the Trigger and Echo pins of the SRF04 or SRF05.

Figure 6.35 Configure the rpi srf node

- Create a **function** node and name it as **Distance**. Enter the following statements into this node (see Figure 6.36). Here, variable **Distance** is loaded with the actual measured distance in centimeters. The first row of the LCD displays a heading, while the second row displays the measured distance. Click **Done**:

```

var Distance = msg.payload;
msg.payload={msgs:[
    {msg:"Distance (cm)"},
    {msg: Distance}
]};
return msg;
  
```

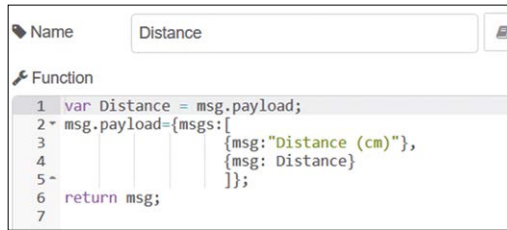


Figure 6.36 Configure the function node

- Create the LCD node as before, join all 3 nodes and click **Deploy**. You should see the distance displayed on the LCD. Place something in front of the sensor and you should see the distance changing (see Figure 6.37)



Figure 6.37 Displaying the distance

Figure 6.38 shows the circuit built on a breadboard (notice in this figure that the author had no 2K resistors and used 2x1K resistors instead).

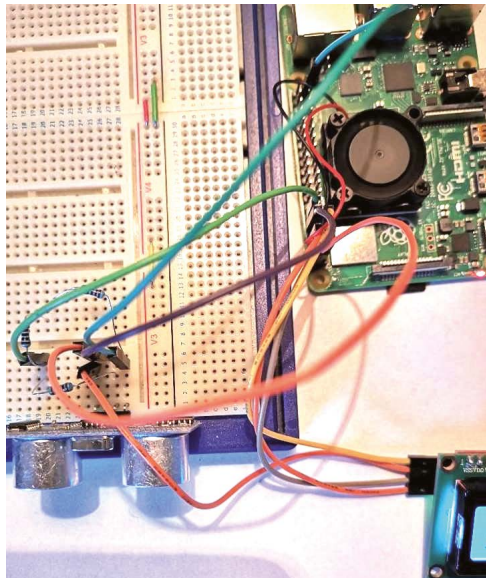


Figure 6.38 The circuit built on a breadboard

6.11 Project 29 – Ultrasonic distance alarm with buzzer

Description: In this project, an ultrasonic distance sensor module is used, as in the previous project. Additionally, an active buzzer is used such that if the distance to any obstacle is less than 20cm then the buzzer is activated. This project can be used in mobile robotic applications or to aid while parking your car.

Aim: The aim of this project is to show how an ultrasonic sensor can be used together with a buzzer in a Node-RED project.

Block diagram: Figure 6.39 shows the project block diagram.



Figure 6.39 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 6.40, where an active buzzer is connected to port pin GPIO 23. The remainder of the circuit is the same as in Figure 6.33.

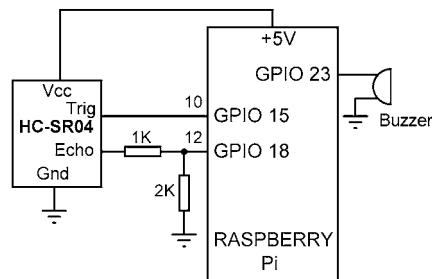


Figure 6.40 Circuit diagram of the project

Node-RED flow program: Figure 6.41 shows the flow program which consists of 3 nodes: a **rpi srf** node to read the distance from the sensor, a **function** node to format the data to control the buzzer, and a **rpio gpio out** node which is connected to the buzzer.



Figure 6.41 Flow program of the project

The steps are as follows:

- Create a **rpi srf** node as in the previous project, but set the Repeat interval to a lower value, e.g. 0.25 second
- Create a **function** node and enter the following statements. If the distance to any obstacle is less than 20cm then the output is set to 1 to activate the buzzer:

```
var Distance = msg.payload;
if(Distance < 20)
    msg.payload = 1;
else
    msg.payload = 0;
return msg;
```

- Create a **rpi gpio out** node and name it as **Buzzer**. Set the **Pin** to GPIO 23, and initialize the pin state to 0
- Join all 3 nodes, and click **Deploy**. Place an object close to the ultrasonic module and you will hear the buzzer sounding. When the object is removed, the buzzer should stop sounding.

6.12 Project 30 – Ultrasonic car parking system with buzzer

Description: In this project, it is assumed that two ultrasonic distance sensor modules are attached to the front and rear of a vehicle. Additionally, an active buzzer is used such that if the distance to any obstacle at the front or rear of the vehicle is less than 20cm then the buzzer is activated. This project can be used to aid while parking your car.

Aim: The aim of this project is to show how two ultrasonic distance sensor modules can be used together with a buzzer in a Node-RED project.

Block diagram: Figure 6.42 shows the project block diagram.



Figure 6.42 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 6.43. One of the ultrasonic sensor modules and the buzzer are connected as in the previous project. The Trig and Echo pins of the second ultrasonic buzzer module are connected to Raspberry Pi port pins GPIO 24 and GPIO 25 respectively (physical pins 18 and 22).

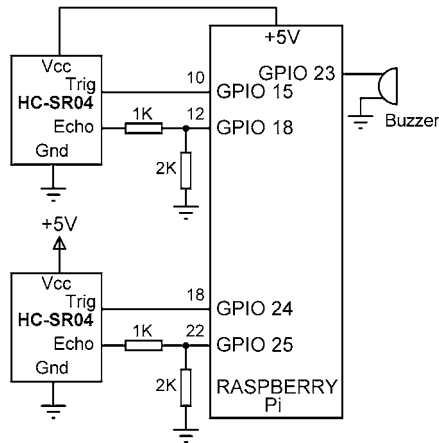


Figure 6.43 Circuit diagram of the project

Node-RED flow program: Figure 6.44 shows the flow program which consists of 5 nodes: 2 **rpi srf** nodes, one for each ultrasonic distance sensor module, a **join** node to join the two outputs into an array, a **function** node to format the output for the buzzer, and a **rpi gpio out** node to control the buzzer.

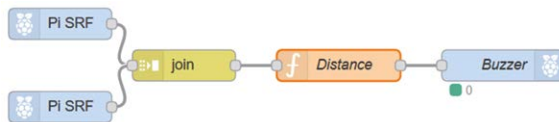


Figure 6.44 Flow program of the project

The steps are as follows:

- Create the two **rpi srf** nodes and set their pins to 10,12 and 18,22. Set the **Repeat** times to 0.25 second in each one
- Create a **join** node and configure as shown in Figure 6.45. This node joins the two inputs into an array that can be manipulated in the function node

Figure 6.45 Configure the join node

- Create a **function** node and enter the following statements inside this node, click **Done**:

```
var Distance1 = msg.payload[0];
var Distance2 = msg.payload[1];
if(Distance1 < 20 || Distance2 < 20)
    msg.payload = 1;
else
    msg.payload = 0;
return msg;
```

- Create a **rpi gpio out** node and set the pin number to GPIO 23
- Join all the nodes as in Figure 6.44, click **Deploy**. Place objects close to either sensor and you should hear the buzzer sounding.

Figure 6.46 shows the circuit built on a breadboard ready for testing.

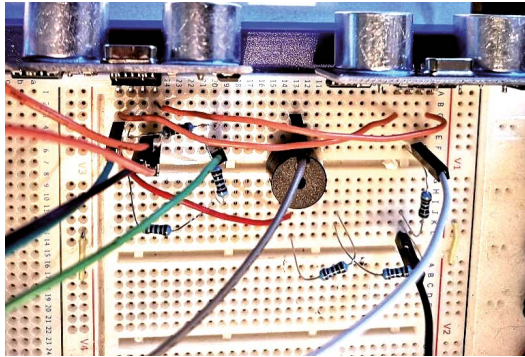


Figure 6.46 Circuit built on a breadboard

6.13 Using a parallel LCD

In the previous projects in this chapter, we have used I²C based LCDs. In these projects, the actual LCD was an HD44780 type parallel LCD, but a small board was plugged at the back of the LCD so that the LCD can communicate using the SDA and SCL I²C lines. Parallel LCDs have the advantages that they are much cheaper than the I²C versions. Their disadvantage is that their control requires at least 6 pins, instead of 2 pins required for the I²C versions. Parallel LCDs also require an external potentiometer to adjust their contrast.

In the remaining parts of this chapter, we will see how an HD44780 type parallel LCD can be used with Node-RED projects.

HD44780 LCD module

HD44780 is one of the most popular alphanumeric LCD modules used in industry and also by hobbyists. This module is monochrome and comes in different sizes. Modules with 8, 16, 20, 24, 32, and 40 columns are available. Depending on the model chosen, the number of rows varies between 1, 2 or 4. The display provides a 14-pin (or 16-pin) connector to a microcontroller. Table 6.1 gives the pin configuration and pin functions of a 14-pin LCD module. Below is a summary of the pin functions:

Pin no	Name	Function
1	VSS	Ground
2	VDD	+ ve supply
3	VEE	Contrast
4	RS	Register Select

5	R/W	Read/Write
6	E	Enable
7	D0	Data bit 0
8	D1	Data bit 1
9	D2	Data bit 2
10	D3	Data bit 3
11	D4	Data bit 4
12	D5	Data bit 5
13	D6	Data bit 6
14	D7	Data bit 7
15	A	Anode (light)
16	K	Cathode (light)

Table 6.1 Pin configuration of HD44780 LCD module

VSS is the 0V supply or ground. The VDD pin should be connected to the positive supply. Although the manufacturers specify a 5V D.C supply, the modules will usually work with as low as 3V or as high as 6V.

Pin 3 is named VEE and this is the contrast control pin. This pin is used to adjust the contrast of the display and it should be connected to a variable voltage supply. A potentiometer is normally connected between the power supply lines with its wiper arm connected to this pin so that contrast can be adjusted.

Pin 4 is the Register Select (RS) and when this pin is LOW, data transferred to the display is treated as commands. When RS is HIGH, character data can be transferred to and from the module.

Pin 5 is the Read/Write (R/W) line. This pin is pulled LOW in order to write commands or character data to the LCD module. When this pin is HIGH, character data or status information can be read from the module.

Pin 6 is the Enable (E) pin which is used to initiate the transfer of commands or data between the module and the microcontroller. When writing to the display, data is transferred only on the HIGH to LOW transition of this line. When reading from the display, data becomes available after the LOW to HIGH transition of the enable pin and this data remains valid as long as the enable pin is at logic HIGH.

Pins 7 to 14 are the eight data bus lines (D0 to D7). Data can be transferred between the microcontroller and the LCD module using either a single 8-bit byte or as two 4-bit nibbles. In the latter case, only the upper four data lines (D4 to D7) are used. 4-bit mode has the advantage that four fewer I/O lines are required to communicate with the LCD.

A simple project is given below for completeness, which illustrates how a parallel LCD can be used in a Node-RED based project.

6.14 Project 31 – Displaying message on a parallel LCD

Description: In this project, a parallel LCD is connected to the Raspberry Pi and the message Hello There is displayed on the LCD.

Aim: The aim of this project is to show how a parallel LCD can be used in a Node-RED based project.

Circuit diagram: Figure 6.47 shows the circuit diagram of the project. The LCD is connected to Raspberry Pi as follows:

LCD Pin	Raspberry Pi GPIO Pin	Raspberry Pi Physical Pin no
D4	GPIO 15	10
D5	GPIO 18	12
D6	GPIO 23	16
D7	GPIO 24	18
RS	GPIO 25	22
E	GPIO 8	24
R/W	GND	6
GND	GND	20
Vcc	+5V	2

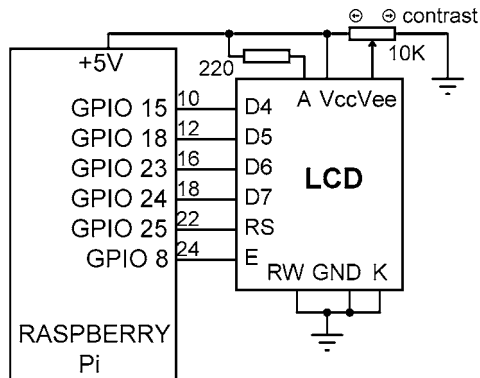


Figure 6.47 Circuit diagram of the project

Notice that a 10K potentiometer must be connected to the VEE (or Vo) pin to set the LCD contrast.

Installing the parallel LCD node: The steps to install the parallel LCD node to Node-RED palette are given below:

- Click **Menu -> Manage** palette and then click **Install**
- Enter **node-red-node-pilcd** into the search pad and click **install**

- You should see a new node named **rpi lcd** in the Node Palette
- The node **rpi lcd** requires to specify where the following LCD pins are connected to:

RS, E, D4, D5, D6, D7

Node-RED flow program: Figure 6.48 shows the flow program which consists of 4 nodes: an **inject** node, 2 **function** nodes, where each one sends text to each line of the LCD, and the **rpi lcd** control node.

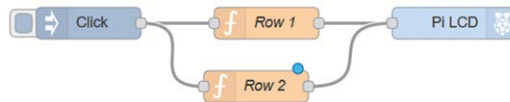


Figure 6.48 Flow program of the project

The steps are as follows:

- Create an **inject** node
- Create two **function** nodes and name them as **Row 1** and **Row 2**. These functions contain the text to be displayed on the LCD. Row 1 sends the text for the top row of the LCD, while Row 2 sends the text for the second row of the LCD. The contents of **Row 1** function is:

```

var first = "1:This is Line 1";
msg.payload = first;
return msg;

```

Similarly, the contents of Row 2 function is:

```

second="2:This is Line 2";
msg.payload=second;
return msg;

```

- Data to the LCD must be sent as strings. For Row 1, start the data with **1:**, for Row 2 start the data with **2:** and so on. String **clr:** clears the screen.

Interested readers can get more information from the following link:

<https://flows.nodered.org/node/node-red-node-pilcd>

- Create the **rpi lcd** node. Configure the node as shown in Figure 6.49.

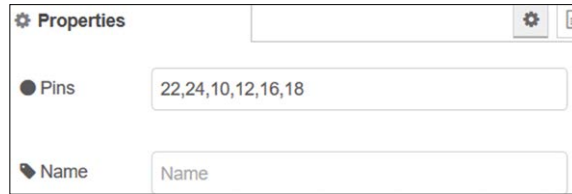


Figure 6.49 Configure node `rpi lcd`

- Join all the nodes and click **Deploy**. Click the button of the **inject** node and you should see the following texts displayed on the LCD:

This is Line 1

This is Line 2

6.15 Summary

In this chapter, we have learned how to use an LCD in Node-RED based Raspberry Pi projects.

In the next chapter, we will be looking at the Analog-to-Digital Converters (ADC) and see how we can use ADCs in Node-RED based projects.

Chapter 7 • Using ADC in Raspberry Pi Node-RED projects

7.1 Overview

Most sensors in real life have analog outputs and such outputs cannot be connected directly to computer input ports. Analog-to-digital converters (ADC) are used to convert these analog signals to digital form so that they can be read by a computer. Unfortunately, Raspberry Pi computer has no built-in ADC ports and it is not possible to interface it to analog outputs unless an external ADC circuit is used.

There are many types of ADCs that can be used in projects. Some important ADC parameters that one should take into account before using one are as follows:

Word length: Most ADCs used to be 8 bits wide, having only 256 quantization levels. Such ADCs are not accurate since only 256 levels are used to represent the input voltage. The general-purpose ADCs nowadays are usually 10 or 12-bits. A 10-bit ADC has 1024 quantization levels, and a 12-bit ADC has 4096 quantization levels. Considering an 8-bit ADC, with a reference voltage of +3.3V, each bit corresponds to 12.89mV. This means that input voltages less than 12.89mV cannot be differentiated. With a 10-bit ADC, each bit represents 3.22mV, and with a 12-bit converter, each bit represents 0.80mV.

In this chapter, we will be using an external ADC on our Raspberry Pi to read data from sensors using Node-RED.

7.2 The MCP3002 ADC

The ADC we will be using in our projects in this Chapter is the MCP3002. This is an 8-pin DIL chip (see Figure 7.1), with the following features:

- 10-bit resolution
- 2 input channels
- Successive approximation converter
- On-chip sample and hold circuit
- SPI interface to host computer
- +2.7 to +5.5V operation
- 200ksps sampling rate (at +5V)
- 5nA standby current

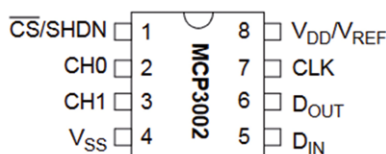


Figure 7.1 MCP3002 pin diagram

MCP3002 incorporates on-chip sample and hold circuitry which is required while converting fast analog input signals. The pin functions are:

CS/SHDN:	Chip select input
CH0:	analog input channel 0
CH1:	analog input channel 1
Vss:	power supply ground
Din:	SPI data input
Dout:	SPI data output
CLK:	SPI clock
Vdd/Vref:	power supply and reference voltage

Installing the MCP3002 node: Before using the MCP 3002 in a Node-RED application, we have to install the MCP3002 node in the Node Palette (the following steps install MCP 3xxx nodes, which includes other ADC chips in the same family). The steps are given below:

- Click **Menu -> Manage palette** and click **Install**
- Enter **node-red-node-pi-mcp3008** in the search pad and click **install**
- When the installation is finished, you should see a new node called **A/D converter**

You can also run the following command from your default Node-RED directory of `~/.node-red`

```
npm install node-red-node-pi-mcp3008
```

We must also enable the SPI interface on our Raspberry Pi 4. The steps are:

- `pi@raspberrypi:~$ sudo raspi-config`
- Move down to **5. Interfacing options**
- Move to **P4 SPI**, click on it and enable SPI interface

Raspberry Pi has the two SPI ports with the following pins:

SPI Port	SPI Function	Raspberry Pi Pin Name	Physical Port number
SPI0	MOSI	GPIO 10	19
SPI0	MISO	GPIO 9	21
SPI0	CLK	GPIO 11	23
SPI0	CE0	GPIO 8	24
SPI0	CE1	GPIO 7	26
SPI1	MOSI	GPIO 20	38
SPI1	MISO	GPIO 19	35
SPI1	CLK	GPIO 21	40
SPI1	CE0	GPIO 18	12
SPI1	CE1	GPIO 17	11

7.3 Project 32 – Voltmeter with LCD output

Aim: The aim of this project is to show how an ADC can be used in a Node-RED project with Raspberry Pi.

A block diagram illustrating the hardware setup. It shows a sequence of components connected by arrows: 'Voltage in' points to an 'ADC' (Analog-to-Digital Converter) chip, which then points to a 'RASPBERRY PI 4' board, which finally points to an 'LCD' display.

Circuit diagram: Figure 7.3 shows the circuit diagram of the project. The MCP 3002 ADC is connected to Raspberry Pi as follows:

The diagram illustrates the connection of an MCP 3002 ADC module to a Raspberry Pi. The MCP 3002 is connected to a 5V supply and ground. Its VDD pin is connected to the 5V supply, and its GND pin is connected to ground. The CLK pin is connected to GPIO 2, Din to GPIO 11, Dout to GPIO 10, CS to GPIO 9, and VSS to ground. The Raspberry Pi is connected to the LCD module via I2C (SDA to GPIO 2, SCL to GPIO 3) and ground.

• 144

Node-RED flow program: Figure 7.4 shows the flow program which consists of 4 nodes: an **inject** node, an **A/D converter** node, **function** node, and an **LCD20x4 I2C** node.

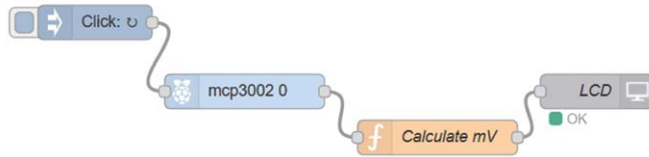


Figure 7.4 Flow program of the project

The steps are as follows:

- Create an **inject** node with an interval of 1 second
- Create an **A/D converter** node and configure it as shown in Figure 7.5. Here, set the **Device** to mcp3002, **Input pin** to A0 since we are using channel CH0, the **Device ID** to CE1, and **SPI bus** to 0, click **Done**

Device	mcp3002
Input pin	A0
Device ID	CE1
SPI bus	0

Figure 7.5 Configure the A/D converter node

- Create a function node and name it as **Calculate mV**, and enter the following statements into this node. Here, the digital value read by the port is converted into physical millivolts by multiplying with $3300.0 / 1024.0$. The floating-point result is then formatted using function **toFixed()** so that only 3 digits after the decimal point are left:

```
var mV = msg.payload * 3300.0 / 1024.0;
var V = mV.toFixed(3) + " mV";
msg.payload={msg: V}
    ];
return msg;
```

- Create an **LCD20x4 I2C** node as in the projects in Chapter 6. Join all the nodes and click **Deploy**.

- Connect the CH0 input of the ADC to an external voltage between 0V and +3.3V. The value of the voltage will be displayed on the LCD in millivolts as shown in the example in Figure 7.6.

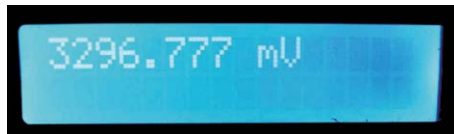


Figure 7.6 Example display of the voltage

7.4 Project 33 – Temperature measurement using analog sensor

Description: In this project, an analog temperature sensor chip is used to measure the ambient temperature and display it every second on the LCD.

Aim: The aim of this project is to show how an analog sensor chip can be interfaced with Raspberry Pi through an ADC in a Node-RED project.

Background information: In this project, the popular TMP36 type analog sensor chip is used. This is a 3-pin chip (Figure 7.7), having the following features:

- 2.7 to 5.5V operation
- $\pm 2^{\circ}\text{C}$ accuracy
- $\pm 0.5^{\circ}\text{C}$ linearity
- -40°C to $+125^{\circ}\text{C}$ measurement range
- Very low current operation

The relationship between the measured temperature and the output voltage is expressed as follows:

$$T = (V_o - 500) / 10$$

Where T is the temperature in $^{\circ}\text{C}$, and V_o is the sensor output voltage in millivolts. For example, if the sensor output voltage is 800mV then the measured temperature is $(800 - 500) / 10 = 30^{\circ}\text{C}$ and so on.

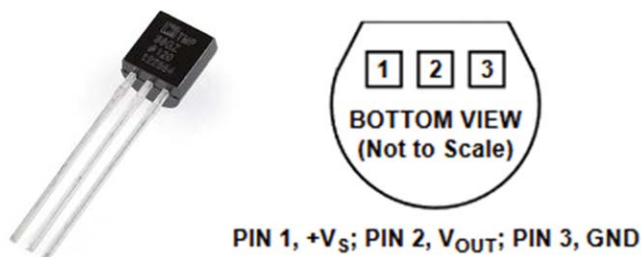


Figure 7.7 TMP36 temperature sensor chip

	Upper 4 bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)																
xxxx0001	(2)																
xxxx0010	(3)																
xxxx0011	(4)																
xxxx0100	(5)																
xxxx0101	(6)																
xxxx0110	(7)																
xxxx0111	(8)																
xxxx1000	(1)																
xxxx1001	(2)																
xxxx1010	(3)																
xxxx1011	(4)																
xxxx1100	(5)																
xxxx1101	(6)																
xxxx1110	(7)																
xxxx1111	(8)																

Figure 7.10 LCD character set

7.5 Project 34 – ON/OFF temperature control

Description: This is an ON/OFF temperature control project of a room. The project consists of a temperature sensor chip, a relay, a heater, an LED, and an LCD. The project controls the heater through the relay so that the temperature in the room is at the desired setpoint value. If the temperature in the room is below the setpoint, then the relay is activated to turn ON the heater. At the same time, the LD is turned ON to indicate that the heater is ON. If on the other hand, the temperature in the room is above the setpoint, then the relay is deactivated so that the heater is turned OFF to bring the room temperature to the required setpoint value. At the same time, the LED is turned OFF to indicate that the heater is OFF. The top row of the LCD shows the setpoint temperature, while the actual room temperature is displayed in the second row of the LCD.

Aim: The aim of this project is to show how an ON/OFF type temperature control project can be developed using Node-RED with a Raspberry Pi.

Background information: An **ON/OFF temperature controller**, also known as a bang-bang controller, will turn OFF the heater whenever the temperature is above a pre-defined setpoint (i.e. the required value). Similarly, if the temperature is below the setpoint then the heater is turned ON. ON/OFF controllers are used for simplicity since there is no need to know the model of the plant to be controlled. The only parameter that needs to be set is the setpoint value. ON-OFF control is suitable when the response delay of the plant to be

controlled is short and the output rate of rise time is small. If precision temperature control is required, then it may be more appropriate to use professional PID (Proportional+Integral+Derivative) type controller algorithms with negative feedback or to use an intelligent fuzzy-based controller. The problem with most professional controllers is that an accurate model of the plant to be controlled is normally required before a suitable control algorithm can be derived. ON/OFF type controller has the disadvantage that the relay used to turn the heater ON and OFF has to operate many times and this may shorten the life of the relay.

Block diagram: Figure 7.11 shows the block diagram of the project.

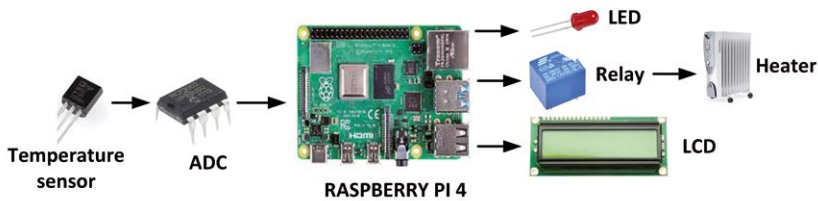


Figure 7.11 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 7.12. Most parts of the circuit are similar to Figure 7.8, but here additionally a relay, a heater, and an LED are connected to the system. Note that some relays operate only with +5V and if you have one of these relays, then you will need to have a +3.3V to +5V logic level converter chip to drive the relay (see the Internet for a suitable converter). The connections between Raspberry Pi and external components are as follows:

Raspberry Pi Port	Component
GPIO 2	LCD SDA
GPIO 3	LCD SCL
GPIO 17	LED anode
GPIO 27	Relay
GPIO 11	MCP 3002 CLK
GPIO 10	MCP 3002 Din
GPIO 9	MCP 3002 Dout
GPIO 7	MCP 3002 CS

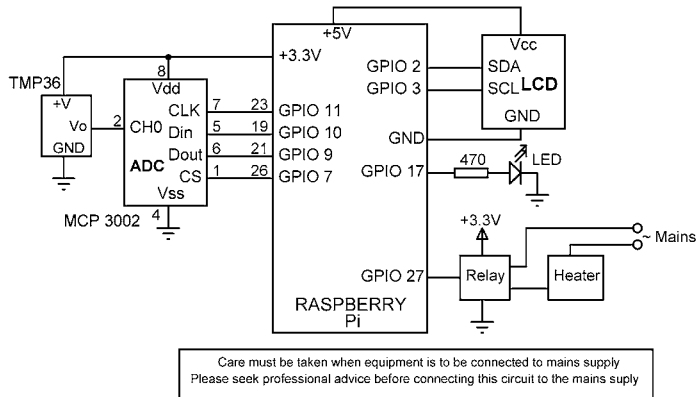


Figure 7.12 Circuit diagram of the project

Node-RED flow program: Figure 7.13 shows the flow program which consists of 6 nodes: an **inject** node, an **A/D converter** node to read the room temperature, a **function** node as the ON/OFF controller, and a **LCD20x4 I2C** node to display the setpoint and the actual temperatures, a **rpi gpio out** node connected to the LED, and another **rpi gpio out** node connected to the relay.

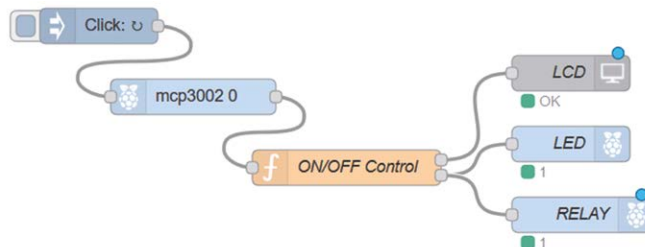


Figure 7.13 Flow program of the project

The steps are as follows:

- Create an **inject** node with an interval of 1 second
- Create an **A/D converter** node and configure it as shown in Figure 7.5
- Create a **function** node with 2 outputs, and name it as ON/OFF Control (see Figure 7.14). Enter the following statements into this node. In this example, the setpoint temperature is set to 22°C. Variable T is loaded with the actual temperature every second. The setpoint temperature is displayed at the top row of the LCD. Similarly, the actual measured temperature is displayed in the second row of the LCD. If the actual measured temperature is less than the setpoint value, then the LED and the relay are turned ON. If on the other hand, the actual measured temperature is equal or greater than the setpoint value, then the LED and the relay are turned OFF. Click **Done**:

```

var setpnt = 22.0;
var DegreeSymbol = String.fromCharCode(0xDF);
var Setpoint = "Setpoint= " + setpnt.toFixed(2) + DegreeSymbol;

var mV = msg.payload * 3300.0 / 1024.0;
var T = (mV - 500.0) / 10.0;
var Temp = "Actual  = " + T.toFixed(2) + DegreeSymbol + "C";

msg.payload={msgs:[
    {msg: Setpoint},
    {msg: Temp}
]};

if(T < setpnt)
{
    var msg1 = {payload: "1"};
    return [msg, msg1];
}
else
{
    var msg2 = {payload: "0"};
    return [msg, msg2];
}

```

Figure 7.14 Configure node function

- Create an **LCD20x4 I2C** node and configure it as before
- Create a **rpi gpio** out node and name it as LED. Set the port name to GPIO 17, and initialize to 0
- Create another **rpi gpio** out node name it as RELAY. Set the port name to GPIO 27, and initialize to 0
- Connect all the nodes as in Figure 7.13 and click **Deploy**. The program should now control the room temperature as required.

Figure 7.15 shows an example display on the LCD where the setpoint temperature is 22°C

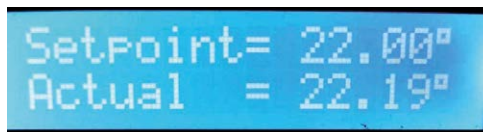


Figure 7.15 Example display

7.6 Summary

In this chapter, we have learned how to use an external analog-to-digital converter chip in a Node-RED based Raspberry Pi project. In the next chapter, we will be installing and using the Dashboard to create live dynamic dashboards in Node-RED projects.

Chapter 8 • The dashboard palette

8.1 Overview

In the last chapter, we have learned how to use an analog-to-digital converter in a Node-RED based Raspberry Pi project. In this chapter, we will be installing and using the Dashboard palette.

The Dashboard palette provides a set of nodes in Node-RED to quickly create a live data dashboard. It is an optional module and not installed by default.

8.2 Installing the dashboard

The instructions to install the Dashboard on Node-RED are as follows:

- Click **Menu -> Manage palette**, and click **Install**
- Enter **node-red-dashboard** and click **install** (see Figure 8.1)

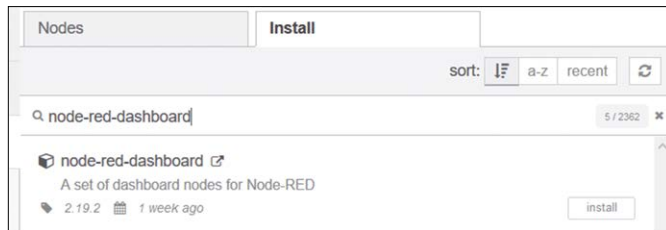


Figure 8.1 Installing the Dashboard

- You will see many new nodes added to the Node Palette under the heading **dashboard** as shown in Figure 8.2

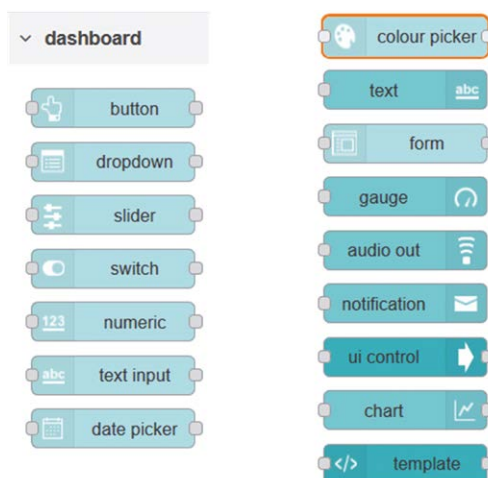


Figure 8.2 The dashboard nodes

We are now ready to use the Dashboard nodes. Some examples are given in the next sections.

8.3 Project 35 – Using a gauge to display the temperature

Description: In the last chapter we have seen how to read and display the temperature on an LCD. In this project, we will read the ambient temperature every second and display it on a gauge node.

Aim: The aim of this project is to show how a gauge node can be used in a Node-RED project

Circuit diagram: The circuit diagram of the project is as in Figure 7.8 where a TMP36 type temperature sensor chip is connected to an MCP 3002 type ADC.

Node-RED flow program: Figure 8.3 shows the flow program which consists of 4 nodes: an **inject** node, an **A/D converter** node, a **function** node, and a **gauge** node.

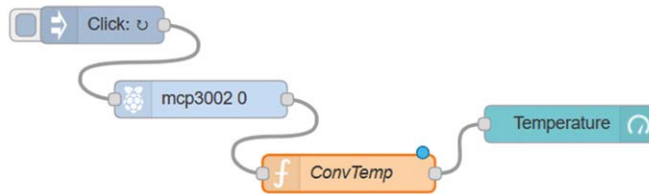


Figure 8.3 Flow program of the project

The steps are as follows:

- Create an **inject** node with an interval of 1 second
- Create an **A/D converter** node as described in Chapter 7
- Create a **function** node and name it as **ConvTemp**. Enter the following statements into this node to read the temperature and convert to degrees Centigrade

```
var mV = msg.payload * 3300.0 / 1024.0;
var T = (mV - 500.0) / 10.0;
msg.payload = T.toFixed(1);
return msg;
```

- Create a **gauge** node and connect all the nodes together as shown in Figure 8.3. Configure the node as follows:
- Double click the **gauge** node

- Click the **Group** field button to add and configure a dashboard. You should see the dashboard group edit form. Configure it as shown in Figure 8.4. Click **Update**

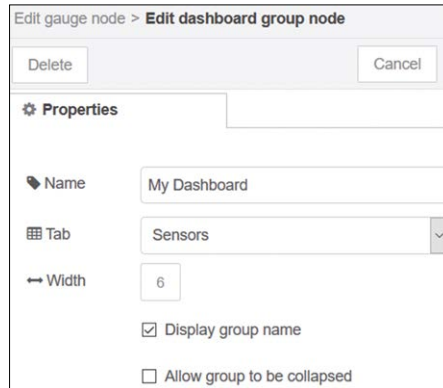


Figure 8.4 Edit the dashboard group name

- Configure the **gauge** node as shown in Figure 8.5. Here, the **Label** is set to Temperature, **Units** to Degrees C, **min** value to 0 and the **max** value to 40. Click **Done**

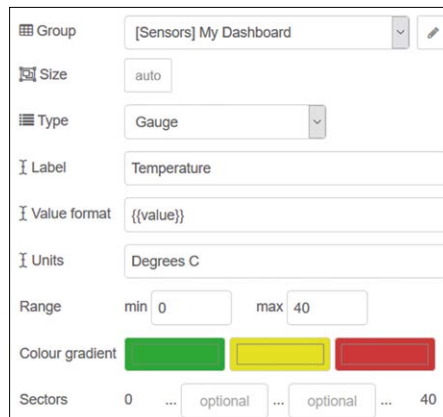


Figure 8.5 Configure the gauge node

- Click the button of the **inject** node. To see the temperature on the gauge display, go to the following site with your browser:

<http://192.168.1.202:1880/ui/>

where 192.168.1.202 is the IP address of your Raspberry Pi. The display in this project is shown in Figure 8.6.

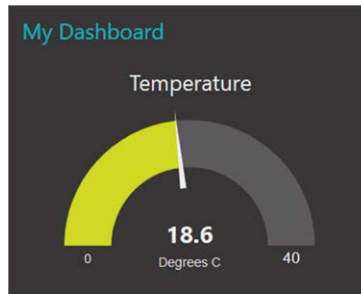


Figure 8.6 Displaying the temperature

The default Tab option is Home, and it allows you to specify which tab of the UI page you will see the UI element on (here our chart). The Name field is the Node-RED node name. The Group field allows you to group UI elements.

8.4 Using a Line Chart to display the temperature

In the last project, we used a gauge to display the ambient temperature. In this section, we will display the temperature using a line chart. The circuit diagram of the project is as in Figure 7.8.

The flow program is the same as in Figure 8.3, but the **gauge** node is replaced with a **chart** node and a **Line chart** is chosen as shown in Figure 8.7.

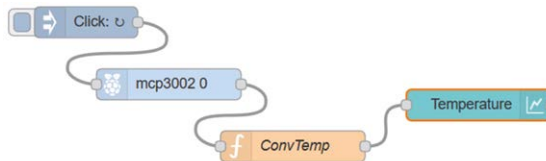


Figure 8.7 Flow program of the example

To configure the chart node, double click on it and configure as shown in Figure 8.8

Group	[Sensors] My Dashboard	
Size	auto	
Label	Temperature	
Type	Line chart	<input type="checkbox"/> enlarge points
X-axis	last 10 minutes	OR 1000 points
X-axis Label	HH:mm:ss	
Y-axis	min 10	max 30
Legend	Show	Interpolate linear

Figure 8.8 Configure the chart node

The display in this example is shown in Figure 8.9.

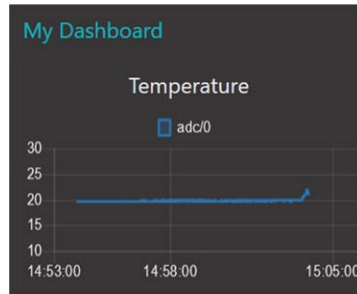


Figure 8.9 Displaying the temperature

8.5 Using a Bar Chart to display the temperature

In the last project, we used a line chart to display the ambient temperature. In this section, we will display the temperature using a bar chart. Since there is only one item to be displayed, the bar chart will have only one column.

The flow program is the same as in Figure 8.3, but the **gauge** node is replaced with a **chart** node and a Bar chart is chosen as shown in Figure 8.10.

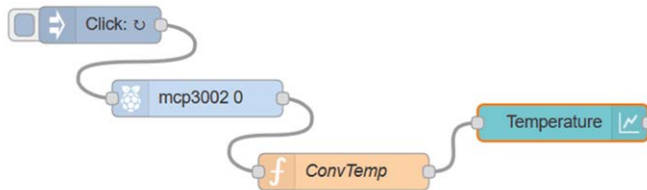


Figure 8.10 Flow program of the example

To configure the chart node, double click on it and configure as shown in Figure 8.11

Figure 8.11 Configure the chart node

The display in this example is shown in Figure 8.12.

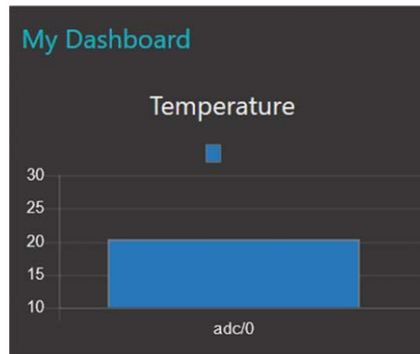


Figure 8.12 Displaying the temperature

You have the other chart options such as Horizontal Bar chart, Pie chart, Polar chart, and Radar chart. Figure 8.13 shows the temperature displayed using a Horizontal bar chart.

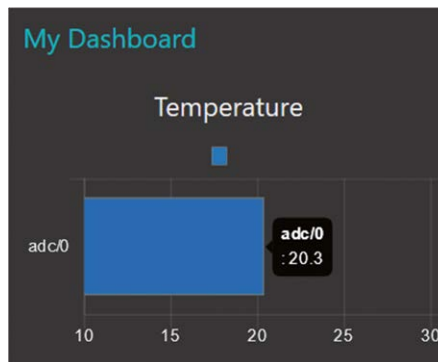


Figure 8.13 Displaying the temperature using a horizontal bar chart

8.6 Project 36 – Using gauges to display the temperature and humidity

Description: In the last examples we displayed only one item using the Dashboard. In this project, we will read the current temperature and humidity from the weather report and then display them using two gauges.

Aim: The aim of this project is to show how two variables can be displayed using two separate gauges.

Node-RED flow program: Figure 8.14 shows the flow program which consists of 5 nodes: an **inject** node, an **openweathermap** node that reads the weather report, a **function** node with 2 outputs that formats the data, and two **gauge** nodes, one to display the temperature and the other one to display the humidity.

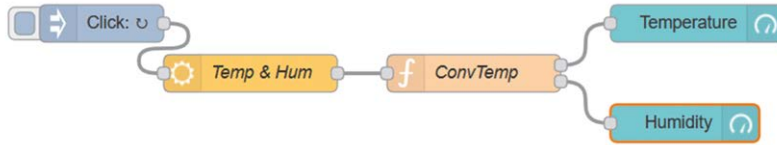


Figure 8.14 Flow program of the project

The steps are as follows:

- Create an **inject** node to repeat every second
- Create an **openweathermap** node and name it as **Temp & Hum**, enter your API key and click **Done**
- Create a **function** node and enter the following statements into this node. Variables T and H extract the temperature and humidity from the current weather report:

```

T = msg.payload.tempc;
H = msg.payload.humidity;
var1 = {payload: T};
var2 = {payload: H};
return [var1, var2];

```

- Create a **gauge** node and set its **Label** to Temperature, **Units** to Degrees C, **min** to 0 and **max** to 30, click **Done**
- Create another **gauge** node and set its **Label** to Humidity, **Units** to %, **min** to 0 and **max** to 100, click **Done**
- Join all the nodes as in Figure 8.14 and click **Deploy**

You should see both the temperature and the humidity displayed in two separate gauges as in Figure 8.15.

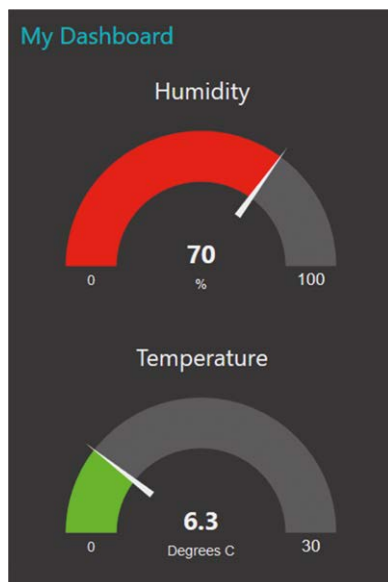


Figure 8.15 Display both the temperature and humidity

8.7 Project 37 – Using multiple gauges

Description: In this project, we will read the local weather forecast and display the following using gauges: minimum temperature, current temperature, maximum temperature, humidity, and atmospheric pressure.

Aim: The aim of this project is to show how multiple gauges can be used in a Node-RED program.

Node-RED flow program: Figure 8.16 shows the flow program which consists of 8 nodes.

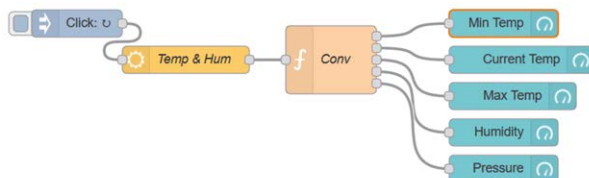


Figure 8.16 Flow program of the project

The steps are as follows:

- Create an **inject** node to repeat every second as in the previous project
- Create an **openweathermap** node as in the previous project
- Create a **function** node with the following statements:

```

Tmin = msg.payload.temp_minc;
Tmax = msg.payload.temp_maxc;
T = msg.payload.tempc;
H = msg.payload.humidity;
P = msg.payload.pressure;
varTmin = {payload: Tmin};
varT = {payload: T};
varTmax = {payload: Tmax};
varH = {payload: H};
varP = {payload: P};
return [varTmin,varT,varTmax,varH,varP];

```

- Create 5 **gauge** nodes for the minimum temperature, current temperature, maximum temperature, humidity, and atmospheric pressure. The configuration of these nodes are as described in the previous project
- Join all the nodes and click **Deploy**.
- Display the Dashboard by entering (enter the IP address of your own Raspberry Pi):

<http://192.168.1.202:1880/ui/>

You should see the displays as in Figure 8.17 (note that the gauges are displayed vertically, and they are cut and pasted horizontally in this figure)



Figure 8.17 Display the gauges

8.8 Project 38 – Using a slider to change LED brightness

Description: In this example, an LED is connected to one of the Raspberry Pi ports. The brightness of the LED is changed by using a slider node to change the duty cycle of the PWM voltage waveform sent to the LED.

Aim: The aim of this project is to show how a slider node can be used.

Circuit diagram: The circuit diagram of the project is the same as in Figure 5.4 where the LED is connected to port pin GPIO 2 through a current limiting resistor.

Node-RED flow program: Figure 8.18 shows the flow program which consists of 3 nodes: a **slider** node to change the duty cycle of the PWM waveform (hence the brightness of the LED), a **rpio gpio out** node to control the LED, and a **gauge** node to display the current value of the duty cycle.

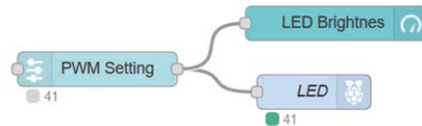


Figure 8.18 Flow program of the project

The steps are as follows:

- Create a **slider** node and name it as **PWM Setting**. Configure this node as shown in Figure 8.19, click **Done**

Figure 8.19 Configure the slider node

- Create a **rpio gpio out** node and name it as **LED**. Set the pin to GPIO 2, and the **Type** to PWM, click **Done**
- Create a **gauge** node with name **LED Brightness** and configure it as shown in Figure 8.20, click **Done**

The screenshot shows the configuration panel for a gauge node in Node-RED. The settings are as follows:

- Group:** [Sensors] My Dashboard
- Size:** auto
- Type:** Gauge
- Label:** LED Brightnes
- Value format:** {{value}}
- Units:** %
- Range:** min 0, max 100

Figure 8.20 Configure the gauge node

- Join all the nodes and click **Deploy**.
- Display the Dashboard by entering (enter the IP address of your own Raspberry Pi):

<http://192.168.1.202:1880/ui/>

- Move the slider at the bottom part of the gauge (see Figure 8.21). You should see the brightness of the LED changing as the slider is moved from 0% to 100% (it is set to 41% in the figure).

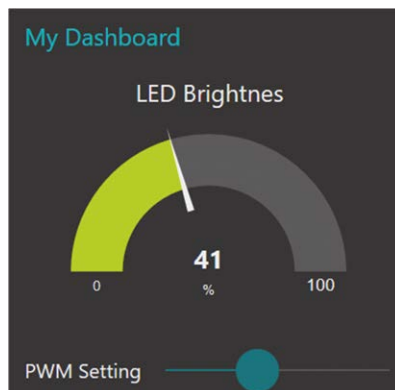


Figure 8.21 Move the slider to change the LED brightness

8.9 Project 39 – Using button nodes to control an LED

Description: In this example, an LED is connected to one of the Raspberry Pi ports as in the previous project. The LED is turned ON and OFF using button nodes from the Dashboard

Aim: The aim of this project is to show how the Dashboard button nodes can be used

Circuit diagram: The circuit diagram of the project is the same as in Figure 5.4 where the LED is connected to port pin GPIO 2 through a current limiting resistor.

Node-RED flow program: Figure 8.22 shows the flow program which consists of 3 nodes: a **button** node to turn ON the LED, a **button** node to turn OFF the LED, and a **rpi gpio out** button connected to the LED.

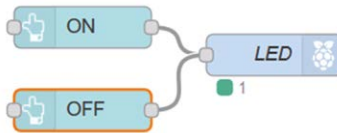


Figure 8.22 Flow program of the project

The steps are as follows:

- Create a **button** node and set its **Label** to **ON**, and the **Payload** to 1 so that the output of the node becomes 1 when clicked in the Dashboard. Configure this node as shown in Figure 8.23, click **Done**

Figure 8.23 Configure the button node

- Create another **button** node and set its **Label** to **OFF**, and the **Payload** to 0 so that the output of the node becomes 0 when clicked in the Dashboard
- Create a **rpi gpio out** node and set the **Pin** to **GPIO**, **Type** to **Digital output**, and initialize to 0
- Join all the nodes and click **Deploy** and display the Dashboard as shown in Figure 8.24. Click **ON** to turn ON the LED, and **OFF** to turn OFF the LED

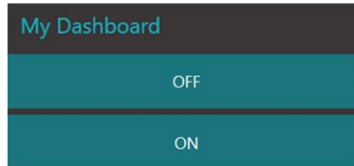


Figure 8.24 Turn the LED ON and OFF

8.10 Project 40 – Using switch and text nodes to control an LED

Description: In this example, an LED is connected to one of the Raspberry Pi ports as in the previous project. The LED is turned ON and OFF by clicking a switch. Additionally, a text is displayed on top of the switch to tell the user what to do

Aim: The aim of this project is to show how the Dashboard switch and text nodes can be used in a project.

Circuit diagram: The circuit diagram of the project is the same as in Figure 5.4 where the LED is connected to port GPIO 2 of Raspberry Pi.

Node-RED flow program: Figure 8.25 shows the flow program which consists of 3 nodes: a **switch** node to turn ON/OFF the LED, a **text** node to display a message, and a **rpi gpio out** button connected to the LED.

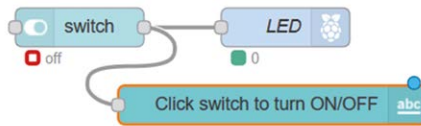


Figure 8.25 Flow program of the project

The steps are as follows:

- Create a **switch** node and set the **On Payload** to 1, and **Off Payload** to 0, as shown in Figure 8.26, click **Done**

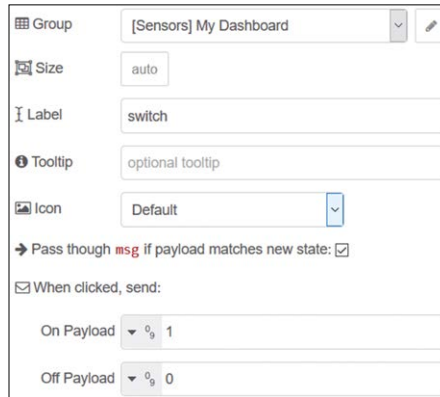


Figure 8.26 Configure the switch node

- Create a **text** node and set the **Label** to Click switch to turn ON/OFF
- Create a **rpi gpio out** node and set the **Pin** to GPIO, **Type** to Digital output, and initialize to 0
- Join all the nodes and click **Deploy** and display the Dashboard as shown in Figure 8.27. Click the switch to turn ON/OFF the LED

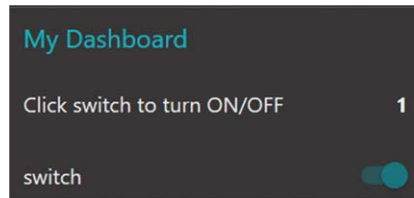


Figure 8.27 Turn the LED ON and OFF

8.11 Project 41 – Talking weather forecast

Description: This is an interesting talking weather project. In this project the local weather report is received using the **openweathermap** node and the data is decoded and then sent to the PC speakers using the Dashboard **audio output** node. As a result, you can hear the weather forecast on your PC speakers. The weather forecast is repeated after 30 seconds when it stops.

Aim: The aim of this project is to show how the Dashboard **audio output** node can be used in a project.

Node-RED flow program: Figure 8.28 shows the flow program which consists of 12 nodes: an **inject** node to start and repeat the weather forecast, an **openweathermap** node to get the local weather forecast, a **function** node that extracts the various local weather parameters and sends them to delay nodes, 7 **delay** nodes that insert delays between the messages, a **join** node that joins the messages, and a Dashboard **audio out**

node that generates sound on the PC speakers.

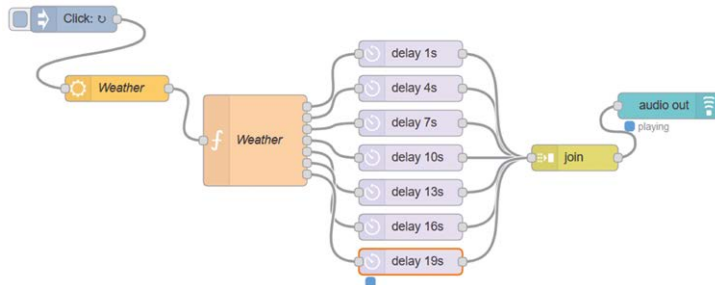


Figure 8.28 Flow program of the project

The steps are as follows:

- Create an **inject** node and set the repeat interval to 54 seconds
- Create an **openweathermap** node and name it as **Weather**. Enter the API key, the local city and country names (see Figure 8.29)

Figure 8.29 Configure the openweathermap node

- Create a **function** node with 7 outputs and name it as **Decode** (see Figure 8.30). Enter the following statements inside this node. Here, L is the location, T is the local temperature in Degrees Centigrade, H is the local humidity in %, P is the local atmospheric pressure in millibars, W is the local wind speed in metres per second:

```
L = msg.payload.location;
T = msg.payload.tempc;
H = msg.payload.humidity;
P = msg.payload.pressure;
W = msg.payload.windspeed;
TT = {payload: "The temperature is "+T+" degrees centigrade"};
HH = {payload: "The humidity is "+H+" percent"};
PP = {payload: "The atmospheric pressure is "+P+" millibars"};
WW = {payload: "The wind speed is "+W+" metres per second"};
```

```

Head = {payload: "This is the weather forecast for " + L + " today"};

if(T < 15)
    MD = {payload: "The weather is very cold and you are advised to
wear a thick coat"};
else
    MD = {payload: "The weather is mild and no need for a thick
coat"};

Nxt = {payload: "The next weather forecast will be in 30 seconds
time"};

return [Head, TT, HH, PP, WW, MD, Nxt];

```

An example weather forecast spoken on the speakers is as follows:

This is the weather forecast for London today. The temperature is 7.1 degrees centigrade. The humidity is 53%. The atmospheric pressure is 1020 millibars. The wind speed is 3.2 meters per second. The weather is very cold and you are advised to wear a thick coat. The next weather forecast will be in 30 seconds time.

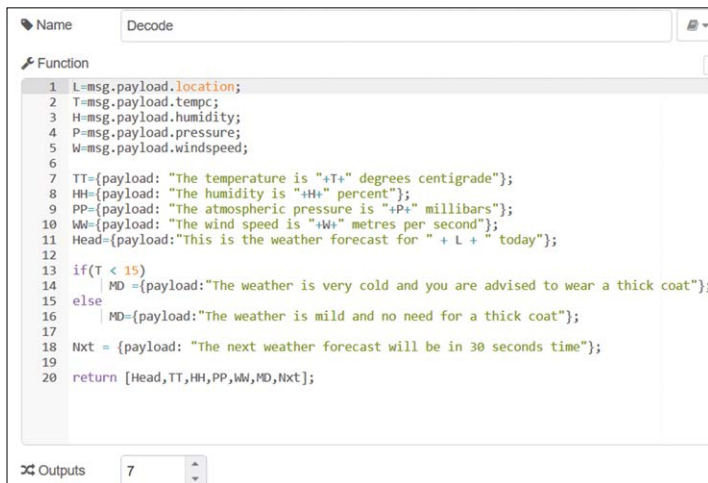


Figure 8.30 Configure node function

- Create a **join** node and configure it as shown in Figure 8.31

Figure 8.31 Configure node join

- Click the **audio out** node in the Dashboard palette, drag and drop in the workspace. Configure this node as shown in Figure 8.32

Figure 8.32 Configure node audio out

- Join all the nodes as shown in Figure 8.28 and click **Deploy**.
- Start your Dashboard (enter your own IP address):

<http://192.168.1.202:1880/ui/>

- Click the button of the **inject** node. You should hear the weather forecast repeating on your speakers. You might have to increase the volume control of your speakers

8.12 Configuring the dashboard

The appearance of the dashboard can easily be configured. Click **Menu -> View -> Dashboard** and you will see the Dashboard window displayed at the right-hand side of the screen. Click the **Theme** tab to set the style as Dark, Light, or Custom. You can also set the Dashboard colour and the font.

8.13 Summary

In this chapter, we have learned how to use some of the Dashboard nodes in projects. In the next chapter, we will be using some of the network nodes to control the devices attached to our Raspberry Pi from a smart mobile phone

Chapter 9 • Wi-Fi UDP/TCP Network-Based projects

9.1 Overview

In the last chapter, we have learned how to use the Dashboard to create some interesting projects. In this chapter, we will be creating Wi-Fi network-based projects.

UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) are two of the most commonly used communications protocols in Wi-Fi-based applications to send and receive data packets over a network. TCP/IP is a reliable protocol that includes handshaking and thus guarantees the delivery of packets to the required destination. UDP, on the other hand, is not so reliable but is a fast protocol. Table 9.1 shows a comparison of the UDP and TCP/IP type communications.

TCP/IP	UDP
Connection-oriented protocol	Connectionless protocol
Slow	Fast
Highly reliable data transmission	Not so reliable
Packets arranged in order	No ordering of packets
20-byte header size	8-byte header size
Error checking and re-transmission	No error checking
Data received acknowledgment	No acknowledgment
Used in HTTP, HTTPS, FTP etc	Used in DNS, DHCP, TFTP etc

Table 9.1 Comparison of UDP and TCP/IP

UDP and TCP/IP protocol based programs are server-client based where one node sends data and the other node receives it and vice-versa. Data is transferred through ports where the server and the clients must use the same port numbers.

Developing UDP and TCP based programs require good programming knowledge. Luckily, Node-RED provides nodes that simplify this task considerably. In this Chapter, we will be developing Node-RED based projects to establish communication between a Raspberry Pi and smart mobile phone.

9.2 Project 42 – Controlling an LED from a mobile phone – UDP based communication

Description: In this example, an LED is connected to one of the Raspberry Pi ports as in the previous Chapter, in Project 40. The LED is turned ON and OFF by sending messages from a mobile phone using the UDP protocol. Valid commands are:

ON turn ON the LED
OFF turn OFF the LED

Aim: The aim of this project is to show how UDP based network communication can be established using Node-RED.

Block diagram: Figure 9.1 shows the block diagram of the project.

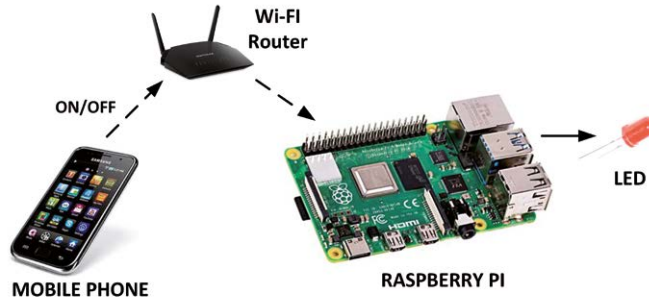


Figure 9.1 Block diagram of the project

Circuit diagram: The circuit diagram of the project is the same as in Figure 5.4 where an LED is connected to port GPIO 2 of Raspberry Pi.

Node-RED flow program: Figure 9.2 shows the flow program which consists of 3 nodes: a **udp in** node to receive UDP packets from the mobile phone, a **function** node to format the received data to control the LED, and a **rpi gpio out** node that is connected to the LED.



Figure 9.2 Flow program of the project

The steps are as follows:

- Create a **udp in** node and configure it as shown in Figure 9.3. Here, the **Port** number is set to 5000, and **Output** is set to string, click **Done**

Figure 9.3 Configure node udp in

- Create a **function** node and name it as ON/OFF. Enter the following statements into this function. Here, the output is set to 1 if the command is ON, and it is set to 0 if the command is OFF. Click **Done**:


```
var md = msg.payload.substr(0, 3);
if(md == "ON")
    msg.payload = 1;
else if(md == "OFF")
    msg.payload = 0;
return msg;
```

- Create a **rpi gpio out** node and set the pin to GPIO 2, name the node as **LED**, set it to digital output, and initialize the pin to 0. Click **Done**
- Join all 3 nodes and click **Deploy**.

The flow program is now complete and it waits to receive UDP commands from a mobile phone to control the LEDs. In this project, an Android-based mobile is used for testing the project. There are many free of charge UDP apps in the Play Store that can be used to send and receive messages. The one used in this project is called **UDP RECEIVE and SEND** by *Wezzi Studios* (v 3.3), as shown in Figure 9.4

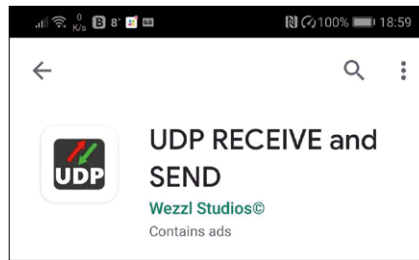


Figure 9.4 UDP RECEIVE and SEND apps

The steps to use this app is as follows:

- Start the apps on your mobile phone
- Set the IP address to the address of your Raspberry Pi (in this project it is 192.168.1.202)
- Set the Port number to 5000
- Enter message **ON** and click SEND UDP MESSAGE as shown in Figure 9.5 and you should see the LED to turn ON

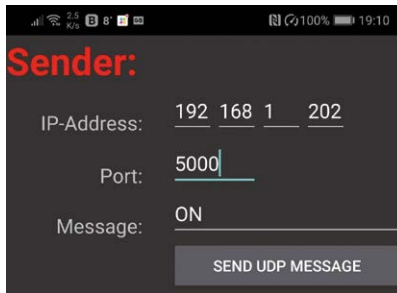


Figure 9.5 Enter ON to turn ON the LED

Enter message **OFF** and click SEND UDP MESSAGE as shown in Figure 9.6 and you should see the LED to turn OFF

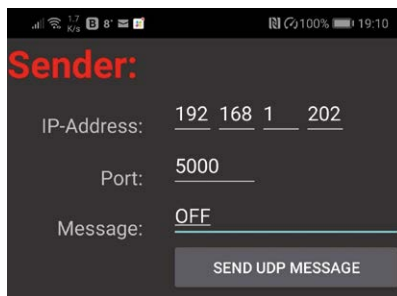


Figure 9.6 Enter OFF to turn OFF the LED

Although in this project we controlled a simple LED, there is no reason why we cannot use for example a relay so that electrical equipment (e.g. a boiler, a lamp, etc) can be controlled remotely.

9.3 Project 43 – Controlling multiple LEDs from a mobile phone – UDP based communication

Description: In the previous project we controlled only one LED. There are applications where we may want to control a number of devices remotely. In this project, we will be controlling 3 LEDs from a mobile phone. Valid commands are:

1=ON	turn ON LED 1
2=ON	turn ON LED 2
3=ON	turn ON LED3
1=OFF	turn OFF LED 1
2=OFF	turn OFF LED 2
3=OFF	turn OFF LED 3

Aim: The aim of this project is to show how multiple devices can be controlled remotely using a Node-RED UDP node.

Block diagram: Figure 9.7 shows the block diagram of the project.

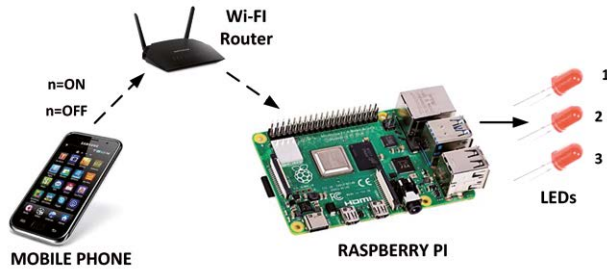


Figure 9.7 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 9.8. The LEDs are connected to GPIO ports GPIO 2, GPIO 3 and GPIO 4.

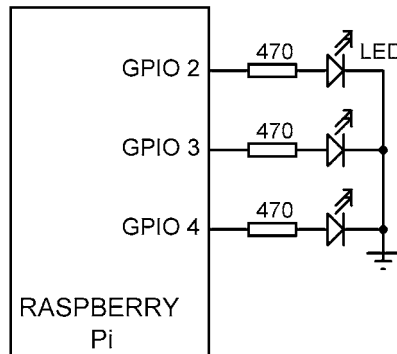


Figure 9.8 Circuit diagram of the project

Node-RED flow program: Figure 9.9 shows the flow program which consists of 5 nodes: a **udp in** node to receive UDP packets from the mobile phone, a function node with 3 outputs to format the received data to control the 3 LEDs, and 3 **rpi gpio out** nodes that are connected to the LEDs.



Figure 9.9 Flow program of the project

The steps are as follows:

- Create a **udp in** node as in the previous project

Create a **function** node with 3 outputs and enter the following statements inside this node:

```
var no = msg.payload.substr(0, 1);
```

```
var md = msg.payload.substr(2, 3);
var var1 = null;
var var2 = null;
var var3 = null;

if(md == "ON")
{
    if(no == "1")
    {
        var1 = {payload: 1};
    }
    else if(no == "2")
    {
        var2 = {payload: 1};
    }
    else if(no == "3")
    {
        var3 = {payload: 1};
    }
}
else if(md == "OFF")
{
    if(no == "1")
    {
        var1 = {payload: 0};
    }
    else if(no == "2")
    {
        var2 = {payload: 0};
    }
    else if(no == "3")
    {
        var3 = {payload: 0};
    }
}

return [var1, var2, var3];
```

- Create 3 **rpi gpio out** nodes with the following pins:

LED1: GPIO 2
LED2: GPIO 3
LED3: GPIO 4

- Join all the nodes as in Figure 9.9 and click **Deploy**.

To test the project, you can use the apps **UDP RECEIVE and SEND** described in the previous project. For example, enter **2=ON** and LED2 should turn ON, enter **2=OFF** and the same LED should turn OFF and so on.

As suggested in the previous project, the LEDs can be replaced with relays to control electrical equipment remotely.

9.4 Project 44 – Sending the temperature and humidity to the mobile phone – UDP based communication

Description: In the previous projects we sent data from the mobile phone to Raspberry Pi. In this project, we will be doing the opposite. i.e. we will be sending data from Raspberry Pi to the mobile phone. The program will wait to receive the command **weather** from the mobile phone. When this command is received, the local weather forecast will be received and the temperature and humidity will be sent to the mobile phone using the UDP protocol. The operation of the program is as follows:

Mobile phone: Sends command **weather** to Raspberry Pi

Raspberry Pi: Gets local weather report and sends temperature and humidity data to mobile phone

Aim: The aim of this project is to show how data can be sent from Raspberry Pi to the mobile phone using the UDP protocol.

Block diagram: Figure 9.10 shows the block diagram of the project.

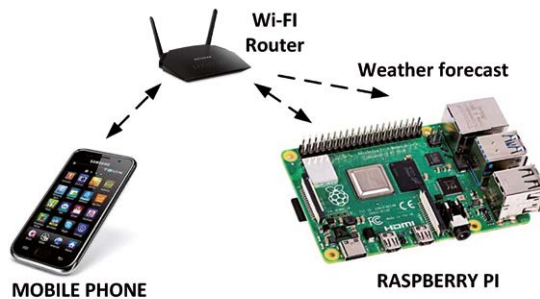


Figure 9.10 Block diagram of the project

Node-RED flow program: Figure 9.11 shows the flow program which consists of 6 nodes: a **udp in** node to receive UDP packets from the mobile phone, a **function** node which checks when command **weather** is received, a **switch** node which activates the **open-weathermap** node to get the local weather report, a **function** node to extract the local temperature and humidity data from the weather report, and a **udp out** node to send the temperature and humidity data to the mobile phone.

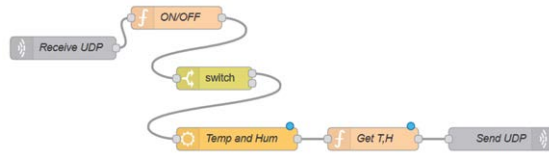


Figure 9.11 Flow program of the project

The steps are as follows:

- Create a **udp in** node as in the previous project and name it as **Receive UDP**
- Create a **function** node and name it as **ON/OFF**. Enter the following statement inside this function will return 1 when command **weather** is received from the UDP:

```

var md = msg.payload;
if(md == "weather")
  msg.payload = 1;
else
  msg.payload = 0;

return msg;

```

- Create a **switch** node that activates the **openweathermap** node when 1 is received. Configure this node as shown in Figure 9.12. Notice that the second output of this node is not used.



Figure 9.12 Configure node switch

- Create an **openweathermap** node and name it as **Temp and Hum**, enter the API key, set your city (London in this project), and country (the UK in this project)
- Create a **function** node and name it as **Get T, H**, enter the following statements inside this function. This function extracts and returns the temperature and humidity from the weather report:

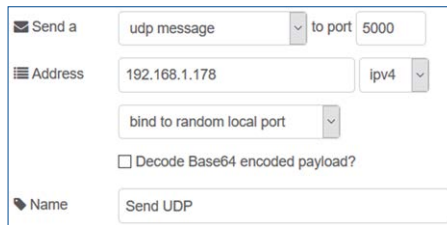
```

T = "Temperature=" + msg.payload.tempc + "C";
H = "Humidity=" + msg.payload.humidity + "%";

```

```
D = T + " " + H;
msg.payload = D;
return msg;
```

- Create a **udp out** node, name it as **SEND UDP**, and configure it as shown in Figure 9.13. Here, the IP address of your mobile phone must be entered into the Address field (in this project this was 192.168.1.178)

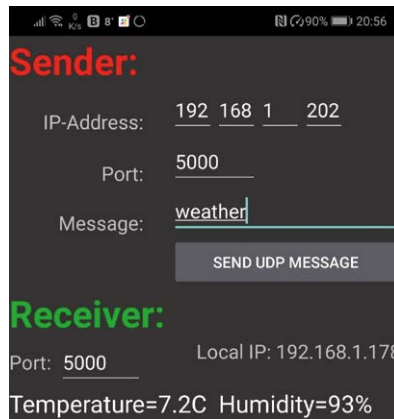


The screenshot shows the configuration for a 'Send a udp message' node in Node-RED. The 'Address' field is set to '192.168.1.178', the 'port' is '5000', and the 'Name' is 'Send UDP'. There are also options for 'bind to random local port' and 'Decode Base64 encoded payload?' which are currently unchecked.

Figure 9.13 Configure node udp out

- Join all the nodes as shown in Figure 9.11 and click **Deploy**

You can test the program using the apps **UDP RECEIVE and SEND** described in the previous projects. Enter the Port number as 5000. Then, enter the command **weather** and click SEND UDP MESSAGE. The local temperature and humidity data will be displayed on your mobile phone as shown in Figure 9.14.



The screenshot shows a mobile app interface with two main sections: 'Sender' and 'Receiver'. The 'Sender' section has fields for 'IP-Address' (192.168.1.202), 'Port' (5000), and 'Message' (weather). There is a 'SEND UDP MESSAGE' button. The 'Receiver' section shows 'Port: 5000', 'Local IP: 192.168.1.178', and the received data: 'Temperature=7.2C Humidity=93%'.

Figure 9.14 Displaying the local temperature and humidity

9.5 Project 45 – Controlling an LED from a mobile phone – TCP based communication

Description: TCP is a reliable connection-based communication protocol. In this example, an LED is connected to one of the Raspberry Pi ports as in the previous Chapter, in Project 42. The LED is turned ON and OFF by sending messages from a mobile phone using the TCP protocol. Valid commands are:

ON turn ON the LED
 OFF turn OFF the LED

Aim: The aim of this project is to show how TCP based reliable network communication can be established using Node-RED.

Block diagram: The block diagram of the project is the same as in Figure 9.1.

Circuit diagram: The circuit diagram of the project is the same as in Figure 5.4 where an LED is connected to port GPIO 2 of Raspberry Pi.

Node-RED flow program: Figure 9.15 shows the flow program which consists of 3 nodes: a **tcp in** node to receive TCP packets from the mobile phone, a **function** node to format the received data to control the LED, and a **rpi gpio out** node that is connected to the LED.



Figure 9.15 Flow program of the project

The steps are as follows:

- Create a **tcp in** node and configure it as shown in Figure 9.16. A TCP node can be configured either as a server or as a client. When configured as a server, the node listens for a connection on a specified port. When configured as a client, the node connects to a specified remote IP address. In this example, the node is configured as a server and the port number is set to 5000

Figure 9.16 Configure node tcp in

- Create a **function** node and name it as ON/OFF as in Project 41. Enter the following statements into this function.

```
var md = msg.payload.substr(0, 3);
if(md == "ON")
    msg.payload = 1;
else if(md == "OFF")
    msg.payload = 0;
return msg;
```


- Create a **rpi gpio out** node and set the pin to GPIO 2 as in Project 41
- Join all 3 nodes and click **Deploy**.

The flow program is now complete and it waits to connect to a TCP client. In this project, an Android-based mobile is used for testing the project as before. There are many free of charge TCP apps in the Play Store that can be used to send and receive TCP messages. The one used in this project is called **Simple TCP Socket Tester** by *Armando Jesus Gomez Parra*, as shown in Figure 9.17

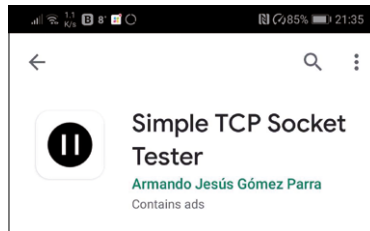


Figure 9.17 Simple TCP Socket Tester apps

The steps to use this app are as follows:

- Start the apps on your mobile phone
- Click tab CLIENT
- Enter the IP address of your Raspberry Pi (in this project it is 192.168.1.202)
- Set the Port number to 5000
- Enter message **ON** and click SEND as shown in Figure 9.18 and you should see the LED to turn ON

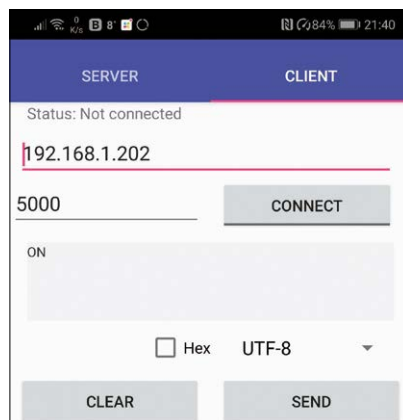


Figure 9.18 Enter ON to turn ON the LED

- Enter message **OFF** and click SEND as shown in Figure 9.19 and you should see the LED to turn OFF

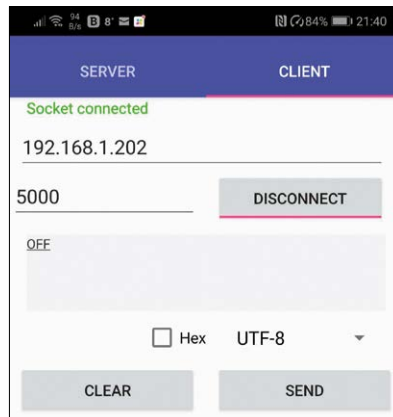


Figure 9.19 Enter OFF to turn OFF the LED

9.6 Project 46 – Controlling Multiple LEDs From a Mobile Phone – TCP Based Communication

Description: In this project, we will control 3 LEDs from a mobile phone as in Project 43, but now we will be using the reliable connection-based TCP protocol.

As before, the valid commands are:

1=ON	turn ON LED 1
2=ON	turn ON LED 2
3=ON	turn ON LED3
1=OFF	turn OFF LED 1
2=OFF	turn OFF LED 2
3=OFF	turn OFF LED 3

Aim: The aim of this project is to show how multiple devices can be controlled remotely using the TCP protocol.

Block diagram: The block diagram of the project is as shown in Figure 9.7.

Circuit diagram: The circuit diagram of the project is as shown in Figure 9.8, where the LEDs are connected to GPIO ports GPIO 2, GPIO 3 and GPIO 4.

Node-RED flow program: Figure 9.20 shows the flow program which consists of 5 nodes: a **tcp in** node to receive TCP packets from the mobile phone, a **function** node with 3 outputs to format the received data to control the 3 LEDs, and 3 **rpi gpio out** nodes that are connected to the LEDs.



Figure 9.20 Flow program of the project

The steps are the same as in Project 43, but the **udp in** node must be replaced with the **tcp in** node. The **tcp in** node must be configured as in Figure 9.16, and the **function** node must be configured as in Project 43.

To test the project, you can use the apps **Simple TCP Socket Tester** described in the previous project. For example, enter **2=ON** and LED2 should turn ON, enter **2=OFF** and the same LED should turn OFF and so on.

9.7 Project 47 – Sending the temperature and humidity to the mobile phone – TCP based communication

Description: This is very similar to Project 44. In this project, we will be sending data from Raspberry Pi to the mobile phone. The program will wait to receive the command **weather** from the mobile phone. When this command is received, the local weather forecast will be received and the temperature and humidity will be sent to the mobile phone using the TCP protocol.

The operation of the program is as described in Project 44.

Aim: The aim of this project is to show how data can be sent from Raspberry Pi to the mobile phone using the TCP protocol.

Block diagram: The block diagram of the project is as in Figure 9.10.

Node-RED flow program: Figure 9.21 shows the flow program which consists of 6 nodes: a **tcp in** node to receive TCP packets from the mobile phone, a **function** node which checks when command **weather** is received, a **switch** node which activates the **open-weathermap** node to get the local weather report, a **function** node to extract the local temperature and humidity data from the weather report, and a **tcp out** node to send the temperature and humidity data to the mobile phone.

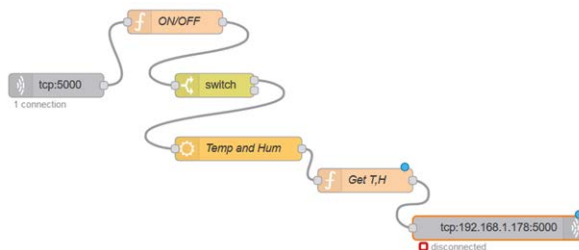


Figure 9.21 Flow program of the project

The steps are the same as in Project 43, except that here the TCP node is used instead of the UDP node. The **tcp in** node at the beginning of the program should be configured as shown in Figure 9.22.

Figure 9.22 Configure the tcp in node

The configuration of the remaining nodes is the same as in Project 43, except the **tcp out** node which is the last node in the flow program. This node should be configured as shown in Figure 9.23, where the **Type** is set to Reply to TCP.

Figure 9.23 Configure the tcp out node

After joining all the nodes as shown in Figure 9.21, click **Deploy**. You can now use the apps **Simple TCP Socket Tester** to test the program. The steps are:

- Start the apps
- Click the CLIENT tab
- Enter the Raspberry Pi IP address
- Enter the port number as 5000
- Enter command weather and click SEND

The current local temperature and humidity values will be displayed on your mobile phone as shown in Figure 9.24

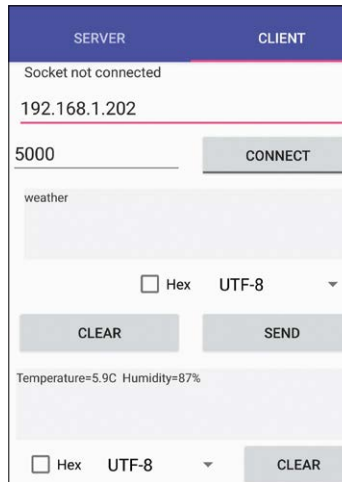


Figure 9.24 Displaying the local temperature and humidity

9.8 Project 48 – Chat program – Mobile phone to Raspberry Pi chat

Description: Chat programs are very common in almost all social media sites. Such programs enable two or more people to communicate in real-time. Chat messages are generally short to enable the participants to respond to each other quickly. Some chat programs also offer video links where the participants can see each other while communicating by text. Most of the chat programs are based on the UDP protocol.

In this project, we develop a simple chat program that enables a person using a mobile phone to chat with another person using a Raspberry Pi. Node-RED Dashboard nodes are used in this program to display the message sent between the two parties.

Aim: The aim of this project is to show how a simple chat program can be developed using the UDP protocol on a Raspberry Pi.

Node-RED flow program: The development of a chat program using Node-RED is relatively easy. What is required is a **udp in** node to receive the messages, a **udp out** node to send out messages, and a tool to display the received and sent messages. Therefore, a chat program is basically a two-way communication between two devices over the Internet.

Figure 9.25 shows the flow program of our chat program. The program consists of 4 nodes: a **udp in** node to receive TCP messages from the mobile phone, a Dashboard **text** node to display the received messages, a Dashboard **text input** node to receive the message to be sent to the mobile phone, and a **udp out** node to send the message to the mobile phone.



Figure 9.25 Flow program of the project

The steps are as follows:

- Create a **udp in** node as shown in Figure 9.26

Figure 9.26 shows the configuration for the **udp in** node. The settings are as follows:

- Listen for:** udp messages
- on Port:** 5000 using ipv4
- Output:** a String
- Name:** Receive UDP

Figure 9.26 Configure the *udp in* node

- Create a Dashboard **text** node and configure as in Figure 9.27

Figure 9.27 shows the configuration for the **text** node. The settings are as follows:

- Group:** [Sensors] My Dashboard
- Size:** auto
- Label:** Received Msg:
- Value format:** {{msg.payload}}
- Layout:** Three 'label value' blocks are shown.

Figure 9.27 Configure the *text* node

- Create a Dashboard **text input** form and configure as in Figure 9.28. Notice that the Delay parameter is set to 0 so that messages are sent out only after the Enter key is pressed (otherwise each key will be sent out as soon as it is pressed)

Figure 9.28 shows the configuration for the **text input** node. The settings are as follows:

- Group:** [Sensors] My Dashboard
- Size:** auto
- Label:** Msg to Send:
- Tooltip:** optional tooltip
- Mode:** text input
- Delay (ms):** 0
- If msg arrives on input, pass through to output:** ☐

Figure 9.28 Configure the *text input* node

- Create a **udp out** node and configure it as shown in Figure 9.29

Figure 9.29 Configure the udp out node

- Join the nodes as in Figure 9.25 and click **Deploy**. To test the program, use the apps **UDP RECEIVE and SEND** as described earlier in this Chapter. Start the Dashboard by entering the following at your web browser (replace the IP address with your own Raspberry Pi IP address). Example output on the Dashboard is shown in Figure 9.30. The corresponding display on the mobile phone is shown in Figure 9.31:

<http://192.168.1.202:1880/ui/>

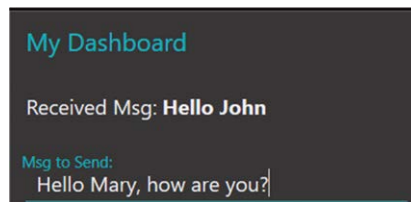


Figure 9.30 Display on the Dashboard

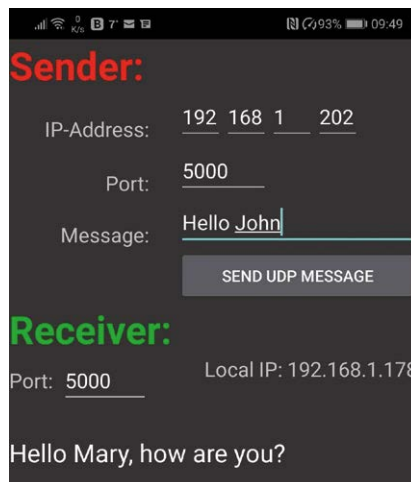


Figure 9.31 Display on the mobile phone

Modified program flow

In the program flow in Figure 9.25, the message sent to the mobile phone (Msg To Send: in Figure 9.30) remains in the text box. As a result, in order to send a new message, we have to delete the existing message. This can be changed so that the message is deleted automatically after 5 seconds of pressing the Enter key. The modified flow program is shown in Figure 9.32.

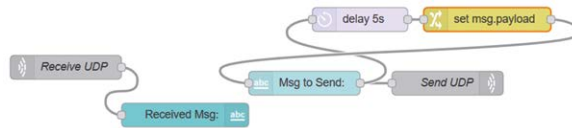


Figure 9.32 Modified flow program

In this modified program, a 5 seconds delay node is inserted at the output of the text. Also, a change node is inserted with no data in it (see Figure 9.33). The result is that the message is deleted after 5 seconds of pressing the Enter key so that a new message can be written.

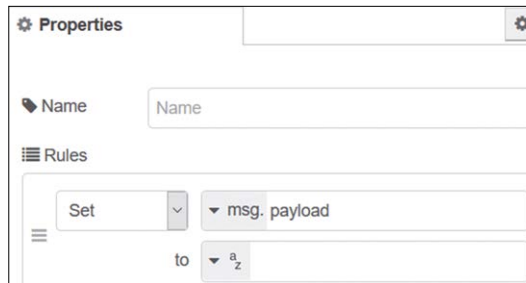


Figure 9.33 Create an empty change node

9.9 Project 49 – Using the ping

Description: Ping is a network utility program used to test the reachability of a computer on a network. Ping measures the round trip time for messages sent from a computer to a destination computer. Basically, a message is sent out and the time for the message to echo back is measured and displayed. Ping works if both computers are available on a network (e.g. Internet or Wi-Fi). In this project, we will send a ping message from our Raspberry Pi to our mobile phone and display the round trip time.

Aim: The aim of this project is to show how the ping utility can be used in a Node-RED application.

Node-RED flow program: The network group of nodes includes a ping node that can be used in an application. The easiest setup is to enter the IP address of the destination computer in node **ping** and connect a debug node to the output of this node. This way, the ping return data will be displayed in the Debug window. In this project, we will use a Dashboard **text** node to display the ping return data dynamically. Figure 9.34 shows the flow program of the project.



Figure 9.34 Flow program of the project

The steps are as follows:

- Create a **ping** node by clicking, dragging and dropping it from the network palette. Double click to configure this node as shown in Figure 9.35. Here, the node is configured to output every 5 seconds. The IP address of the author's mobile phone is given as the **Target** (you should choose your own IP address). Note that you can also enter a valid web address, e.g. www.bbc.co.uk

Figure 9.35 Configure the ping node

- Create a **text** node by clicking, dragging and dropping it from the Dashboard. Configure this node as shown in Figure 9.36

Figure 9.36 Configure the text node

- Join the two nodes and click **Deploy**. Start your dashboard. You should see the round trip time as shown in Figure 9.37

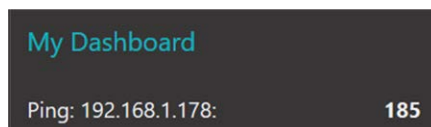


Figure 9.37 Example display of the round trip time

9.10 Summary

In this chapter, we have learned how to use UDP and TCP nodes in Node-RED based applications to communicate with a mobile phone.

In the next chapter, we will be looking at storage nodes and learn how we can store data in a file.

Chapter 10 • Storage Nodes

10.1 Overview

In the last chapter, we have learned how to use Node-RED nodes for UDP and TCP based network communication. In this chapter, we will be looking at the important topic of how to store data in files on a Raspberry Pi SD card, using Node-RED nodes.

Data on Raspberry Pi is normally stored on the SD card which is also loaded with the operating system. The default user folder is `/home/pi`. In this chapter, we will be creating files in the default directory and then store data in these files.

10.2 Project 50 – Storing timestamped temperature and humidity data in a file

Description: In this project, we will be reading the local weather report every minute using the **openweathermap** node and then extract the temperature and the humidity data. This data will then be stored in a file called **temphum.txt** on the Raspberry Pi SD card together with the time that the data was extracted.

Node-RED flow program: Figure 10.1 shows the flow program which consists of 5 nodes: an **inject** node which repeats every minute, an **openweathermap** node which gets the local weather report, a **function** node which formats the data, a **change** node which specifies the name of the file, and a **file** node which stores (appends) the data into the specified file.

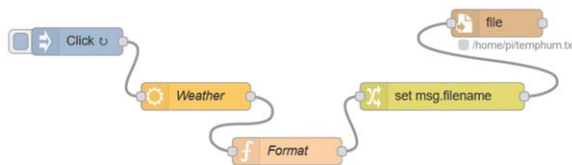


Figure 10.1 Flow program of the project

The steps are as follows:

- Create an **inject** node, name it as **Click**, and set it to repeat every minute
- Create an **openweathermap** node and name it as **Weather**. Enter the API key
- Create a **function** node, name it as **Format**, and enter the following statements inside this node. Variable **first** is initialized to 0 as a context variable. This is done so that we can insert a heading to the data the first time the function is run. Thereon, the first is non-zero and the actual data is stored in the file. The local time is extracted and stored in a variable called **Tim** which is stored in the file with every data item.

```

var first = context.get('first') || 0;
if(first == "0")
{
    context.set('first',1);
    All = " Time    Temperature  Humidity";
}

var LocalTime = new Date();
var Tim = LocalTime.toLocaleTimeString();
Temp = msg.payload.tempc;
Hum = msg.payload.humidity;
if(first != "0")
    All = Tim + "      " + Temp + "C          " + Hum + "%";
msg.payload = All
return msg;

```

- Create a **change** node and set **msg.filename** to the name of the file where we want the data to be stored, i.e. /home/pi/temphum.txt. Note that we could have set the filename in the node **file** as well. If msg.filename is used then the file will be closed after every write. Configure this node as shown in Figure 10.2

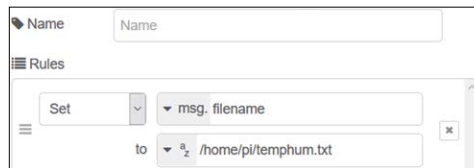


Figure 10.2 Configure node change

- Create a **file** node and configure it as shown in Figure 10.3. Notice that the **Action** is set to Append to file, and a new line is to be added after each record

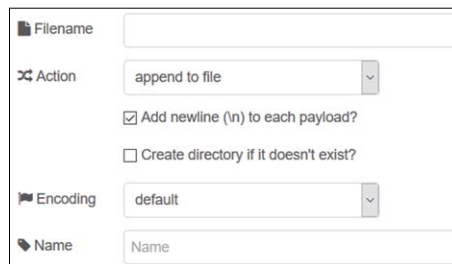


Figure 10.3 Configure node file

- Join all the nodes as shown in Figure 10.1, and click **Deploy**.
- Click the button of the **inject** node. A new file called temphum.txt will be created in folder /home/pi and the local temperature and humidity data will be stored in the file every minute.

Figure 10.4 shows the files (using command **ls**) in the default user folder `/home/pi`. Notice that file `temphum.txt` is in this folder. The contents of file `temphum.txt` (command **cat temphum.txt**) are shown in Figure 10.5.

```
pi@raspberrypi:~ $ ls
book      Downloads  Music      Pictures  temphum.txt
Desktop   MagPi      node_modules Public    Templates
Documents mu_code    package-lock.json Scratch   Videos
pi@raspberrypi:~ $
```

Figure 10.4 Files in folder `/home/pi`

```
pi@raspberrypi:~ $ cat temphum.txt
Time      Temperature Humidity
16:40:30  7.2C       87%
16:41:30  7.2C       93%
16:42:30  7.2C       93%
16:43:30  7.2C       93%
16:44:31  7.2C       93%
16:45:30  7.1C       93%
16:46:31  7.2C       93%
16:47:30  7.2C       93%
16:48:30  7.1C       93%
16:49:30  7.1C       93%
16:50:30  7.1C       93%
16:51:30  7.1C       93%
16:52:30  7.1C       93%
```

Figure 10.5 Contents of file `temphum.txt`

10.3 Project 51 – Reading contents of a file

Description: In the previous project we have seen how to store data in a file. In this project, we will read the contents of the file that we created in the previous project.

Node-RED flow program: Node **file in** is used to read the contents of a file either as a string or as a binary buffer. Figure 10.6 shows the flow program which consists of 3 nodes: an **inject** node, a **file in** node to read the contents of the file, and a **debug** node that displays the contents of the file.



Figure 10.6 Flow program of the project

The steps are as follows:

- Create an **inject** node, name it as **Click**
- Create a **file in** node and configure it as shown in Figure 10.7. The **Filename** is set to `/home/pi/temphum.txt`, and the **Output** is set to a string

Figure 10.7 Configure the file in node

- Create a **debug** node and set its **Output** to debug window. Join the nodes and click **Deploy**
- Click the button of the **inject** node. You should see the contents of the file displayed in the Debug window as shown in Figure 10.8. Notice here that all the contents of the file are displayed and the text is not aligned properly.

Figure 10.8 Contents of the file displayed in the Debug window

- Now, double click on the **debug** node and change the **Output** to the system console. Click the button of the inject node. The system console is the Raspberry Pi command mode display and you should see that the contents of the file are displayed correctly as shown in Figure 10.9

```
To find more nodes and example flows - go to h
Starting as a systemd service.
16 Dec 20:53:57 - [info] [debug:23b139c2.ac37b
Time      Temperature  Humidity
16:40:30      7.2C      87%
16:41:30      7.2C      93%
16:42:30      7.2C      93%
16:43:30      7.2C      93%
16:44:31      7.2C      93%
16:45:30      7.1C      93%
16:46:31      7.2C      93%
16:47:30      7.2C      93%
16:48:30      7.1C      93%
```

Figure 10.9 Displaying the file on the system console

- The file can be read in different formats. Double click on node **file in** and change the **Output** to a msg per line. Also, change the **debug** node to output to the Debug window. Now, click the button of the **inject** node and you should see the contents of the file displayed on the Debug window as shown in Figure 10.10

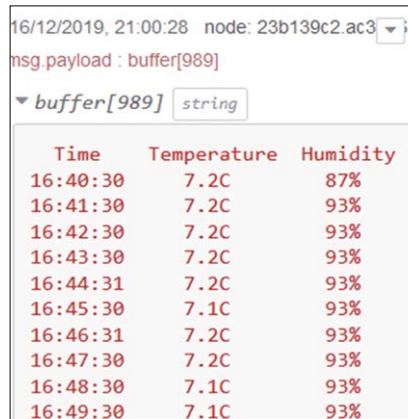
```
"16:40:30 7.2C 87%"
16/12/2019, 20:57:46 node: 23b13
msg.payload : string[28]
"16:41:30 7.2C 93%"
16/12/2019, 20:57:46 node: 23b13
msg.payload : string[28]
"16:42:30 7.2C 93%"
16/12/2019, 20:57:46 node: 23b13
msg.payload : string[28]
"16:43:30 7.2C 93%"
16/12/2019, 20:57:46 node: 23b13
msg.payload : string[28]
"16:44:31 7.2C 93%"
```

Figure 10.10 Displaying the file contents in msg per line mode

- The file can also be displayed in binary format by setting the **Output** of node **file into** a single Buffer object. In this mode, the output is shown in Figure 10.11. This display may not seem to have much meaning. Click on the numbers and you should see the contents of the file displayed correctly as shown in Figure 10.12.

```
16/12/2019, 21:00:28 node: 23b139c2.ac37b6
msg.payload : buffer[989]
▶ [ 32, 32, 84, 105, 109, 101, 32,
32, 32, 32 ... ]
```

Figure 10.11 Displaying the file contents in single Buffer object mode



Time	Temperature	Humidity
16:40:30	7.2C	87%
16:41:30	7.2C	93%
16:42:30	7.2C	93%
16:43:30	7.2C	93%
16:44:31	7.2C	93%
16:45:30	7.1C	93%
16:46:31	7.2C	93%
16:47:30	7.2C	93%
16:48:30	7.1C	93%
16:49:30	7.1C	93%

Figure 10.12 Displaying the file correctly in the Debug window

In summary, reading a file has four options:

utf-8 string: the entire file is converted into a string and read and presented as a text string, and the file cannot be processed before it is completed. Here, the data must contain text data.

one msg per line: the data is read and presented as lines of text, and the file can be processed while reading it

a single Buffer object: the data is read as sequence of raw bytes and the complete data is read, it cannot be processed before it is completed

a stream of buffers: the data is read and presented as a sequence of raw bytes, and it can be processed while reading it

We can easily convert the binary data into string format using function `toString()`. For example, in the following function code, we can set the **file in** format to **a stream of buffers** and convert the data into string format:

```
var data = msg.payload;
srng = data.toString();
msg.payload = srng;
return msg;
```

10.4 Project 52 – Reading a file line by line

Description: In the previous project we have seen how to read the contents of a file using the **file in** node. The problem was that the entire contents of the file were read and therefore it was not possible to process the file while it was being read. In this project, we will read the file created in Project 50 and extract and display the temperature field only.

Node-RED flow program: Node **file in** is used to read the contents of the file as in the

previous project. Figure 10.13 shows the flow program which consists of 5 nodes: an **inject** node, a **file in** node to read the contents of the file, a **delay** node to output one message per second, a **function** node to extract the temperature field of the data, and a **debug** node that displays the extracted data in the Debug window.

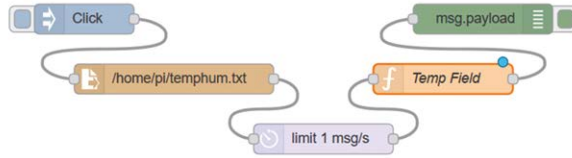


Figure 10.13 Flow program of the project

The steps are as follows:

- Create an **inject** node, name it as **Click**
- Create a **file in** node and set the **Filename** to `/home/pi/temphum.txt`, and the **Output** to a msg per line
- Create a **delay** node set its **Action** to Rate limit with 1 msg per second. Configure this node as shown in Figure 10.14

```

pi@raspberrypi:~ $ ls
book      Downloads  Music      Pictures  temphum.txt
Desktop   MagPi      node_modules Public    Templates
Documents mu_code    package-lock.json Scratch   Videos
pi@raspberrypi:~ $
  
```

Figure 10.14 Configure the delay node

- Create a **function** node named **Temp Field** and enter the following statements inside this node. This node will extract and output the temperature field of the data:

```

T = msg.payload.substr(12,4);
msg.payload = T;
return msg;
  
```

- Create a **debug** node to display the output in the Debug window
- Join the nodes and click **Deploy**. Click the button of the **inject** node. You should see the temperature displayed in the Debug window as shown in Figure 10.15

16/12/2019, 22:17:32	node: 5df26784.abb36
msg.payload : string[4]	
" 7.2"	
16/12/2019, 22:17:33	node: 5df26784.abb36 <input type="button" value="v"/>
msg.payload : string[4]	
" 7.2"	
16/12/2019, 22:17:34	node: 5df26784.abb36
msg.payload : string[4]	
" 7.2"	
16/12/2019, 22:17:35	node: 5df26784.abb36
msg.payload : string[4]	
" 7.2"	
16/12/2019, 22:17:36	node: 5df26784.abb36
msg.payload : string[4]	
" 7.2"	

Figure 10.15 Displaying the temperature

10.5 Summary

Files in Raspberry Pi are normally stored on the SD card that also stores the operating system. In this chapter, we have learned how to use file nodes to write to and also read from files.

In the next chapter, we will be looking at the nodes that are used for serial communication, and develop projects that use serial communication.

Chapter 11 • Serial communication

11.1 Overview

In the last chapter, we have learned how to write and read from files using Node-RED, and we have developed several projects on this topic. This chapter is about serial communication where we will learn how two devices can communicate using serial communication. Serial communication has been one of the earliest methods of establishing communication between two devices. Nowadays, this type of communication is not used much, especially after the invention of USB.

In its simplest form, serial communication between two devices requires only 3 wires: TX (or TXD, transmit), RX (or RXD, receive), and GND (ground). Figure 11.1 shows the simplest form of serial communication between two devices where the TX and RX lines are crossed between the devices.

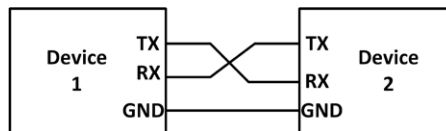


Figure 11.1 Serial communication lines

In serial communication, data bytes are broken down into bits and these bits are sent and received with the correct timings. Data is sent or received as frames where a frame consists of the following bits:

- Start bit
- 7 or 8 data bits
- Optional parity bit
- Stop bit

The parity bit is used for one-bit error checking and this bit is not used very often. Data is normally sent with 8 bits. Therefore, a data frame consists of 10 bits, where a device that wishes to communicate with another device sends out a start bit, followed by 8 data bits, and a stop bit.

The bit timing, also known as the Baud rate, is very important in serial communication and both sides must have exactly the same Baud rates. Typical Baud rates are 9600, 19200, 38400, and so on. The Baud rate effectively determines the number of bytes that can be sent in a second. For example, at 9600 Baud we can send or receive 960 bytes (or characters) per second.

Older serial communications devices used $\pm 12\text{V}$ as the signal levels, where -12 was known as Mark and +12 as Space. These signal levels were specified by the standard RS232 protocol. Two types of connectors were used in RS232 applications: 25-way D-type connector (you can see these connectors on very old computers), and 9-way D-type connector (usually found on old laptops). Nowadays, most microcontrollers operate with either +5V or

+3.3V. Therefore, serial communication signal levels have changed to be compatible with the present-day microcontroller inputs. Sometimes the new levels are also called to be TTL compatible. It is important to be careful and not to connect an RS232 based line to a microcontroller input as the high voltage will damage the microcontroller circuitry.

In this chapter, we will be using the serial in and serial out nodes provided by Node-RED to communicate with the external world over the Raspberry Pi serial port.

11.2 Project 53 – Communicating with Arduino over serial line

Description: In this project, we use an Arduino Uno and a Raspberry Pi. An analog temperature sensor is connected to one of the analog inputs of the Arduino. The Arduino reads the ambient temperature every 5 seconds and sends it to the Raspberry Pi over the serial line. The Raspberry Pi reads the temperature from the serial line and displays it in the Debug window.

Aim: The aim of this project is to show how the serial communication can be established using the Node-RED serial in node.

Background information: The Arduino computer can be programmed to use either its dedicated hardware serial communication port (GPIO port pins 0 and 1) or any of its port pins can be used in what is known as software serial communication. In software serial communication, all the timing is done in software. The only disadvantage of using software instead of hardware in serial communication is that the maximum Baud rate is rather limited. In this application, we shall be using 9600 as the Baud rate, and there are no timing problems at this low Baud rate.

Raspberry Pi computers have two built-in hardware UARTs: a PL011 and a mini UART. These are implemented using different hardware blocks, so they have slightly different characteristics. Since both are 3.3V devices, extra care must be taken when connecting to other serial communication lines. On Raspberry Pis which are equipped with Wireless/Bluetooth modules (e.g. Raspberry Pi 3, Zero W, 4, etc), the PL011 UART is by default connected to the Bluetooth module, while the mini UART is the primary UART with the Linux console on it. In all other models, the PL011 is used as the primary UART. By default, `/dev/ttyS0` refers to the mini UART and `/dev/ttyAMA0` refers to the PL011. The Linux console uses the primary UART which depends on the Raspberry Pi model used. Also, if enabled, `/dev/serial0` refers to the primary UART (if enabled), and if enabled, `/dev/serial1` refers to the secondary UART

By default, the primary UART (serial0) is assigned to the Linux console. Using the serial port for other purposes requires this default configuration to be changed. On startup, systemd checks the Linux kernel command line for any console entries and will use the console defined therein. To stop this behaviour, the serial console setting needs to be removed from the command line. This is easily done by using the `raspi-config` utility by selecting option **5** (Interfacing Options) and then **P6** (Serial) and select **No**. Exit `raspi-config` and re-start your Raspberry Pi. You should now be able to access the serial port (Don't forget to re-enable the console setting after you finish).

In Raspberry Pi 3 and 4, the serial port (/dev/ttyS0) is routed to two pins GPIO 14 (TXD) and GPIO 15 (RXD) on the header. Models earlier than model 3 uses this port for Bluetooth. Instead, a serial port is created in software (/dev/ttyS0).

We can easily search for the available serial ports by entering the following console command:

```
dmesg | grep tty
```

The last line in the output below indicates that the console is enabled on serial port ttyS0.
console [ttyS0] enabled

In this book, we are using Raspberry Pi 4. The serial port is: /dev/ttyS0. If you are using earlier than model 3, use the serial port named: /dev/ttyAMA0

Block diagram: Figure 11.2 shows the block diagram of the project, where the temperature sensor is connected to the Arduino Uno.

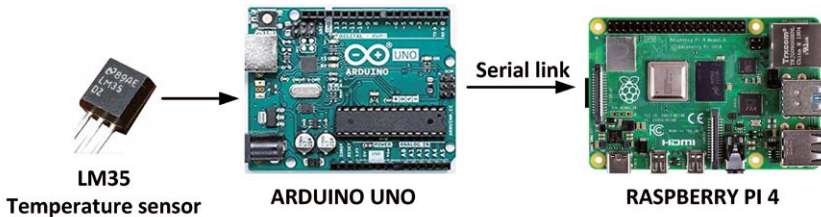


Figure 11.2 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 11.3. In this project, an LM35DZ type analog temperature chip is used and its output pin is connected to analog input A0 of the Arduino Uno. This is a 3-pin device operating with +5V, and its pin configuration is shown in Figure 11.3. The output voltage of LM35DZ is directly proportional to the temperature and is given by the relationship:

$$T = V_o / 10$$

Where, T is the measured temperature in degrees Centigrade, and Vo is the output voltage of the sensor in millivolts. For example, if the output voltage is 200mV, then the measured voltage is 20°C and so on.

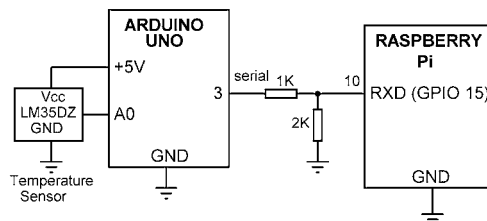


Figure 11.3 Circuit diagram of the project

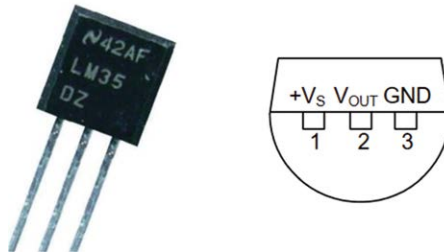


Figure 11.4 LM35DZ pin configuration

The serial connection between the Arduino Uno and Raspberry Pi are as follows. Notice here that GPIO pins 2 and 3 of the Arduino Uno are used for serial communication (pin 2 for RX and pin 3 for TX):

Raspberry Pi	Raspberry Pi Physical Pin No	Arduino Uno
GPIO 14 (TXD) – Not used	8	-
GPIO 15 (RXD)	10	3
Ground	6	GND

Raspberry Pi TXD output is not used in this project. The digital output high voltage of the Arduino Uno is +5V and as a result, we cannot connect the TX output of Arduino Uno directly to the Raspberry Pi RXD input. A resistive potential divider circuit is used to lower the voltage from +5V to +3.3V as shown in Figure 11.3.

Arduino Uno Program: The Arduino program (File: **ArduinoTemp**) listing is shown in Figure 11.5. At the beginning of the program, port pins 2 (not used) and 3 are defined as the software serial communication port lines. Inside the setup routine, the Baud rate is set to 9600. The remainder of the program runs in the main program loop. Here, analog data is received from port A0, is stored in variable sensor as a digital value. This value is then converted into physical voltage in millivolts and stored in floating-point variable mV. The temperature is then calculated by dividing this number by 10 and storing in floating-point variable T. Built-in function `dtostrf()` is used to convert the floating-point value to a string so that it can be displayed by the Raspberry Pi. This function has the following parameters:

```
dtostrf(floating variable, min string width, number of vars after dec point,
character buffer)
```

After converting the temperature value into a string, it is sent through the serial line TX to the Raspberry Pi every 5 seconds. The data is sent in the following format:

$$T = 273.15 + t, \text{ mmC}$$

```

/*-----
 * This program received the ambient temperature from a
 * LM35DZ tye analog temperature sensor connected to pin
 * A0 of the Arduino Uno. The temperature is sent to
 * port 3 in serial form at 9600 Baud
 *
 * File: ArduinoTemp
 *-----*/
#include <SoftwareSerial.h>
SoftwareSerial MySerial(2, 3);           // RX, TX
float mV, T;
char Temp[10];

void setup()
{
    MySerial.begin(9600);
}

void loop()
{
    int sensor = analogRead(A0);         // Read temperature
    mV = sensor * 5000.0 / 1024.0;       // in mV
    T = mV / 10.0;                       // in Centigrade
    dtostrf(T, 4,2,Temp);                // Convert to string
    MySerial.print("T=");                // Display T=
    MySerial.print(Temp);                // Display Value
    MySerial.println("C");               // Display C
    delay(5000);                         // Wait 5 secs
}
    
```

Figure 11.5 Arduino Uno program

We can use the Arduino IDE Serial Monitor to confirm that the program is working as expected, and the temperature is converted into Degrees Centigrade correctly.

Node-RED flow program: Remove the serial console setting as described earlier in this Chapter. Figure 11.6 shows the flow program which consists of just 2 nodes: a **serial in** node which receives serial data, and a **debug node** that is used to display the received data.



Figure 11.6 Flow program of the project

The steps are as follows:

- Create a **serial in** node and configure as shown in Figure 11.7 (Click the pen after Serial Port), click **Update** and **Done**. The **Serial Port** is set to /dev/ttyS0, **Baud rate** is set to 9600, 8 data bits, no parity, and 1 stop bit (these settings must be same as the settings of the transmitting side)

Serial Port: /dev/ttyS0

Settings: Baud Rate: 9600, Data Bits: 8, Parity: None, Stop Bits: 1

Input: Optionally wait for a start character of [], then Split input on the character [] and deliver as ascii strings

Output: Add character to output messages []

Request: Default response timeout 10000 ms

Figure 11.7 Configure the node serial in

- Create a **debug** node, join the nodes and click **Deploy**. You should see the temperature received from the Arduino Uno displayed in the Debug window as shown in Figure 11.8

```

17/12/2019, 18:54:59 node: 563c03b2.5d1cbc
msg.payload : string[10]
  » "T=20.51C"

17/12/2019, 18:55:04 node: 563c03b2.5d1cbc
msg.payload : string[10]
  » "T=20.51C"

17/12/2019, 18:55:09 node: 563c03b2.5d1cbc
msg.payload : string[10]
  » "T=20.51C"
  
```

Figure 11.8 Temperature displayed in the Debug window

Modified flow program 1

We can modify the flow program given in Figure 11.6 so that the received temperature is displayed using a Dashboard **text** node. The new flow program is shown in Figure 11.9

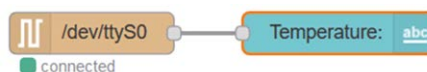


Figure 11.9 Modified flow program

Set the **Label** field of your **text** node to string **Temperature:** You should start your Dashboard. Figure 11.10 shows the temperature displayed with the **text** node.

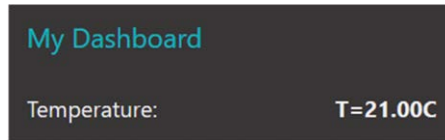


Figure 11.10 Displaying the temperature using a text node

Modified flow program 2

We can modify the flow program again so that the received temperature is spoken on the PC speakers by using a Dashboard audio out node. The new flow program is shown in Figure 11.11.



Figure 11.11 Modified flow program

The steps are as follows:

- Create a **serial in** node as before
- Create a **function** node and enter the following statements inside this function. If for example, the temperature is 20.00°C, then this function will send the text **The temperature now is twenty point zero degrees centigrade** to node audio out so that it can be heard on the PC speakers

```
var T = msg.payload.substr(2, 5);
md = {payload: "The temperature now is " + T + "Degrees Centigrade"};
return [md];
```

- Create an **audio out** node and tick box: Play audio when window not in focus
- Join the nodes, click **Deploy** and start your Dashboard. You should hear the current temperature spoken through your PC speakers

11.3 Project 54 – Receiving serial data – Receiving GPS data

Description: In this project, a GPS receiver module is connected to the serial input of the Raspberry Pi. The **serial in** node is used to receive and display the GPS sentences in the Debug window.

Aim: The aim of this project is to show how serial data can be received using node **serial in**.

Background information: GPS receivers receive geographical data from the GPS satellites and they provide accurate information about the position of the user on Earth. These satellites circle the Earth at an altitude of about 20,000 km and complete two full orbits every day. In order for a receiver to determine its position, it is necessary for the receiver to communicate with at least 3 satellites. Therefore, if the receiver does not have a clear view of the sky then it may not be possible to determine its position on Earth. In some applications, external antennas are used so that even weak signals can be received from the GPS satellites.

The data sent out from a GPS receiver is in text format and is known as the NMEA Sentences. Each NMEA sentence starts with a \$ character and the values in a sentence are separated by commas. Some of the NMEA sentences returned by a GPS receiver are given below:

\$GPGLL: This sentence returns the local geographical latitude and longitude

\$GPRMC: This sentence returns the local geographical latitude and longitude, speed, track angle, date, time, and magnetic variation.

\$GPVTG: This sentence true track, magnetic track, and the ground speed.

\$GGGA: This sentence returns the local geographical latitude and longitude, time, fix quality, number of satellites being tracked, horizontal dilution of position, altitude, the height of geoid, and DGPS data

\$GPGSV: There are 4 sentences with this heading. These sentences return the number of satellites in view, satellite number, elevation, azimuth, and SNR.

In this project, the GPS Click board (www.mikroe.com) is used. This is a small GPS receiver (see Figure 11.12) which is based on the LEA-6S type GPS. This board operates with +3.3V and provides two types of outputs: I²C or serial output. In this project, the default serial output is used which operates at 9600 Baud rate. An external dynamic antenna can be attached to the board in order to improve its reception for indoor use or in for use in places where there may not be a clear view of the sky.



Figure 11.12 GPS Click board

Figure 11.13 shows the complete list of the NMEA sentences output from the GPS Click board every second.

```
$GPGLL,5127.3917,N,00003.13141,E,10534.00,A,A*67
$GPRMC,05305.00,A,5127.35909,.0003,13148,E,0.030,,270919,,A*7E
$GPVTG,.T,.M,0030,N,0.055,K,A*20
$GGGA,105305.00,5127.35909,N,00003.13148,E,1.09,1.18,46.5,M,45.4,M,,*66
$GPRSA,A,3,01.32,08,28,18,03,22,14,11,,2,12,1,18,1,76*06
$GPGSV,4,1,13,01,7,304,40,03,40,224,31,08,38,165,32,10,05,054,*77
$GPGSV,4,2,13,11,83,217,3,14,39,094,24,17,17,314,22,18,73,091,41*76
$GPGSV,4,3,13,22,63,219,33,24,1,002,,27,05,150,,28,30,284,28*7F
$GPGSV,4,4,13,32,34,063,35*4E
```

Figure 11.13 NMEA sentences output from the GPS Click board

GPS Click board is a 2x8 pin dual-in-line module and it has the following pin configuration (pin 1 is the top left pin of the module):

- | | |
|------------------|-------------------|
| 1: No connection | 16: No connection |
| 2: Reset | 15: No connection |
| 3: No connection | 14: TX |
| 4: No connection | 13: RX |
| 5: No connection | 12: SCL |
| 6: No connection | 11: SDA |
| 7: +3.3V | 10: No connection |
| 8: GND | 9: GND |

In serial operation, only the following pins are required: +3.3V, GND, TX. In this project, an external dynamic antenna is attached to the GPS Click board as it was used indoors.

Circuit diagram: Figure 11.14 shows the circuit diagram of the project. The TX serial output pin of the GPS Click board is connected to RXD input of the Raspberry Pi. Since both TX and RXD pins operate with +3.3V, there is no need to lower the voltage for the Raspberry Pi input.

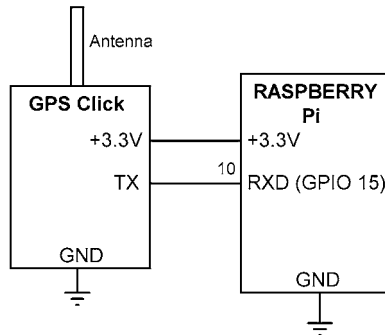


Figure 11.14 Circuit diagram of the project

Node-RED flow program: Remove the serial console setting as described earlier in this Chapter. Figure 11.15 shows the flow program which consists of just 2 nodes: a **serial in** node which receives serial GPS data, and a **debug node** that is used to display the received data.

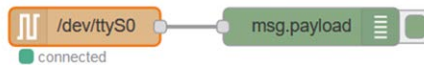


Figure 11.15 Flow program of the project

Configure the serial in node and the debug node exactly as in the previous project. You should see the NMEA sentences displayed in the Debug window as shown in Figure 11.16. Notice that only some of the NMEA sentences are shown in this Figure.

```

17/12/2019, 22:17:07 node: fa4d9c5d.7a2b28
msg.payload : string[52]
▶
"$GPGLL,5127.36351,N,00003.12720,E,221707.00,A,0.098,N,0.181,K,A*2A"

17/12/2019, 22:17:07 node: fa4d9c5d.7a2b28
msg.payload : string[68]
▶
"$GPRMC,221707.00,A,5127.36340,N,00003.12720,E,0.0,A,0.098,N,0.181,K,A*2A"

17/12/2019, 22:17:07 node: fa4d9c5d.7a2b28
msg.payload : string[35]
▶ "$GPVTG,,T,,M,0.098,N,0.181,K,A*2A"

17/12/2019, 22:17:07 node: fa4d9c5d.7a2b28
msg.payload : string[74]
▶ "$GPGGA,221707.00,5127.36340,N,00003.12720,E,1,00,0.0,M,0.0,0.0,0000.0,M,0.0,0.0,0000.0,A"

17/12/2019, 22:17:07 node: fa4d9c5d.7a2b28
msg.payload : string[62]
▶ "$GPGSA,A,3,25,31,14,23,26,16,05,21,02,1,00"

```

Figure 11.16 Displaying the NMEA sentences in the Debug window

\$GPGLL is one of the commonly used NMEA sentences. This sentence is displayed in Figure 11.16 as follows:

```
$GPGLL,5127.37032,N,00003.12782,E,221918.00,A,A*61
```

The fields in this sentence can be decoded as follows:

GLL	Geographic position, latitude and longitude
5127.37032	Latitude 51 deg, 27.3702 min. North
00003.12782	Longitude 0 deg, 3.12782 min. East
221918	Fix taken at 22L19L18 UTC
A	Data active (or V for void)
*61	checksum data

Notice that the fields are separated by commas. The validity of the data is shown by letters A or V in the data, where A shows that the data is valid, and V indicates that the data is not valid.

11.4 Project 55 – Receiving GPS data – Extracting the latitude and longitude

Description: This project is similar to the previous project, but here the latitude, its direction, longitude, and its direction are extracted and displayed in the Debug window.

Aim: The aim of this project is to show how the latitude and longitude values can be extracted from an NMEA sentence. The \$GPGLL sentence is used in this example which includes both the latitude and longitude information.

Circuit diagram: The circuit diagram of the project is the same as in Figure 11.14.

Node-RED flow program: Remove the serial console setting as described earlier in this Chapter. Figure 11.17 shows the flow program which consists of 4 nodes: a **serial in** node which receives serial GPS data, a **switch** node to extract only the \$GPGLL sentence from the NMEA sentences, a **function** node to extract the latitude and longitude from the \$GPGLL, and a **debug** node which is used to display the data in the Debug window.

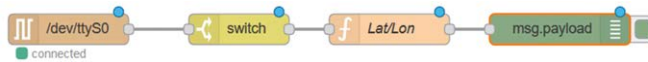


Figure 11.17 Flow program of the project

The steps are as follows:

- Configure the **serial in** node as before
- Create a **switch** node and configure it as shown in Figure 11.18. This node extracts only the \$GPGLL sentences



Figure 11.18 Configure the switch node

- Create a **function** node and name it as **Lat/Lon**. Enter the following statements inside this node. This node extracts the latitude and longitude information and their directions:

```
var nmea=msg.payload.split(",");
var lat=nmea[1];
var latdir = nmea[2];
var lon = nmea[3];
var londir = nmea[4];

T = "Lat="+lat+latdir+" Lon="+lon+londir;
msg.payload=T;
return msg;
```

- Create a **debug** node to display the result in the Debug window. Click Deploy. You should see the latitude, longitude, and directions displayed as shown in Figure 11.19.

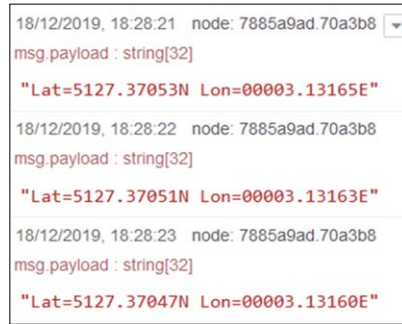


Figure 11.19 Displaying the latitude and longitude information

Modified program

The flow program given in Figure 11.17 displays the latitude and longitude as degrees and minutes. For example, Lat = 5127.3610N means that the latitude is 51 degrees and 27.3610 minutes and it is in the Northern Hemisphere. Similarly, Lon = 00003.12748E means that the longitude is 0 degrees 03.12748 minutes east of Greenwich (where the zero degrees longitude is). In most applications, the latitude and longitude are represented as decimal degrees and if the point is in the Southern Hemisphere (i.e. direction is S), then a negative sign is added to the value. Similarly, if the point is on the west side of Greenwich (i.e. direction is W), then a negative sign is added to the value.

In this section, we will modify the displayed values so that they are displayed as degrees with the correct signs.

The process is as follows. Consider the example where the Lat = 5127.3610S:

- Take the integer part of the value and divide by 100 to give 51
- Take away 51x100 from the original value to find the minutes value, 5127.3610 - 51x100 = 27.361
- Divide the minutes' value by 60 to convert to degrees, 27.361/60 = 0.45601
- Add to the degrees value found earlier, 51 + 0.45601 = 51.45601 degrees
- Since the direction is W, insert a negative sign, -51.45601 degrees

Node-RED flow program: The flow program is the same as in Figure 11.17, where only the contents of the **function** node changes. The flow program is repeated in Figure 11.20 below so that the exported flow program can be loaded from Figure 11.20.



Figure 11.20 Flow program of the project

Enter the following statements inside the **function** node:

```
var nmea=msg.payload.split(",");
var lat=nmea[1];
var latdir = nmea[2];
var lon = nmea[3];
var lonDir = nmea[4];
//
// Process Latitude
//
var Lat = Number(lat);
var Latint = parseInt((parseInt(Lat)) / 100);
var mins = Lat - 100*Latint;
var degrees = mins / 60;
var LatValue = (Latint + degrees).toFixed(5, 2);
if(latdir == "S")
    LatValue = -LatValue;
//
// Process Longitude
//
var Lon = Number(lon);
var Lonint = parseInt((parseInt(Lon)) / 100);
mins = Lon - 100*Lonint;
degrees = mins / 60;
var LonValue = (Lonint + degrees).toFixed(5, 3);
if(lonDir == "W")
    LonValue = -LonValue;

msg.payload="Lat="+LatValue+" Lon="+LonValue;
return msg;
```

Figure 11.21 shows the latitude and longitude in degree format.



Figure 11.21 Displaying latitude and longitude in degree format

11.5 Project 56 – Displaying our location on a map

Description: In this project we receive and decode the GPS coordinates as in Project 55 with the latitude and longitude converted into degrees. We then plot our location on a map using the Dashboard node called **worldmap**.

Aim: The aim of this project is to show how our location can be plotted on a map using a node called **worldmap**.

Circuit diagram: The circuit diagram of the project is the same as in Figure 11.14, where a GPC Click board receives our position data and passes it to the Raspberry Pi.

Node-RED flow program: In this project, we use a Dashboard node called **worldmap**. You must install this node in your Node-RED before it can be used. The steps are:

- Click **Menu -> Manage palette**, and click **Install**
- Enter **node-red-contrib-web-worldmap** and click **install**
- When the installation is complete, you should see the node worldmap in your Node Palette.

The flow program of this project is shown in Figure 11.22. It consists of 4 nodes: a **serial in** node to receive the GPS data in real-time, a **switch** node to extract the \$GPGLL sentences, a **function** node to extract the latitude and longitude in degrees format, and a **worldmap** node to show our position on a map.

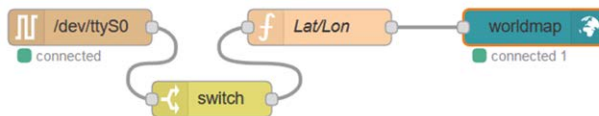


Figure 11.22 Flow program of the project

The steps to develop the flow program are:

- Create a **serial in** node as before
- Create a **switch** node as in Figure 11.18
- Create a **function** node, name it as **Lat/Lon** and enter the following statements inside this node. Notice how the latitude and longitude should be specified by variable **msg.payload**:

```

var nmea=msg.payload.split(",");
var lat=nmea[1];
var latdir = nmea[2];
  
```

```

var lon = nmea[3];
var londir = nmea[4];
//
// Process Latitude
//
var Lat = Number(lat);
var Latint = parseInt((parseInt(Lat)) / 100);
var mins = Lat - 100*Latint;
var degrees = mins / 60;
var LatValue = (Latint + degrees).toFixed(5, 2);
if(latdir == "S")
    LatValue = -LatValue;
//
// Process Longitude
//
var Lon = Number(lon);
var Lonint = parseInt((parseInt(Lon)) / 100);
mins = Lon - 100*Lonint;
degrees = mins / 60;
var LonValue = (Lonint + degrees).toFixed(5, 3);
if(londir == "W")
    LonValue = -LonValue;

msg.payload = {"name":"Home", "lat":LatValue, "lon":LonValue};
return msg;

```

- Create a **worldmap** node and configure it as shown in Figure 11.23. Make the following settings (assuming you are in the UK):

Base map:	UK OS Opendata
Zoom:	16 (this can be changed)
Lock map:	False
Auto-pan:	Enable
Coordinates:	Degrees
Lock Zoom:	False
Right Click:	Disable
Graticule:	Visible

By setting **Auto-pan** to Enable, we enabling panning of the map. By setting the **Lock Zoom** to False we can change the zoom level of the map by clicking the + and – signs on the top left-hand side of the map. You should try changing these parameters and see their effects.

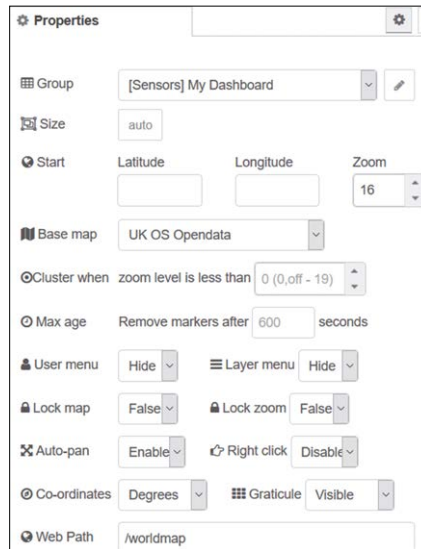


Figure 11.23 Configure node worldmap

- Join all the nodes as in Figure 11.22 and click **Deploy**.
- Start the Dashboard by entering your own IP:

<http://192.168.1.202:1880/ui/>

- You should see your position marked on the map as shown in figure 11.24. Notice that here the map is zoomed to show the street level.

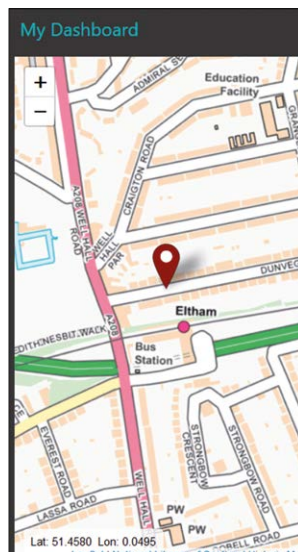


Figure 11.24 Your position shown on the map

You can change the zoom level easily. Figure 11.25 shows the map after zooming out.

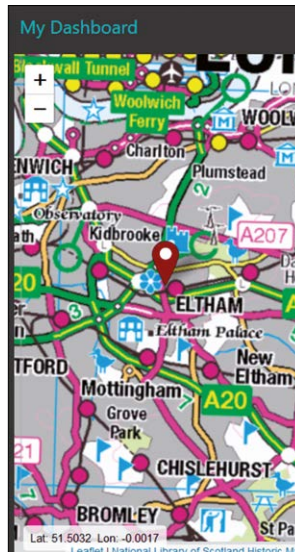


Figure 11.25 Zooming out the map

You can also change the Base map easily by double-clicking on the **worldmap** node. Figure 11.26 shows our position with the Base map set to terrain.

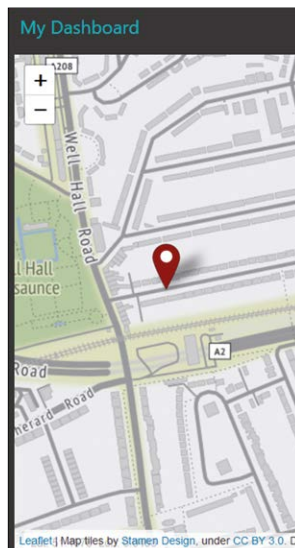


Figure 11.26 Base map set to Terrain

Node **worldmap** includes many features that are not covered in this project as this topic is beyond the scope of this book. Interested readers can look at the following web site for very detailed information:

<https://flows.nodered.org/node/node-red-contrib-web-worldmap>

11.6 Project 57 – Sending out serial data to arduino

Description: In this project, we will receive the weather report using node **openweathermap** and then send the temperature and humidity readings to the Arduino Uno every 5 seconds on a serial line. The Arduino will display the temperature at the top row and the humidity at the bottom row of an LCD.

Aim: The aim of this project is to show how serial data can be outputted from Raspberry Pi using the **serial out** node.

Block diagram: The block diagram of the project is shown in Figure 11.27.

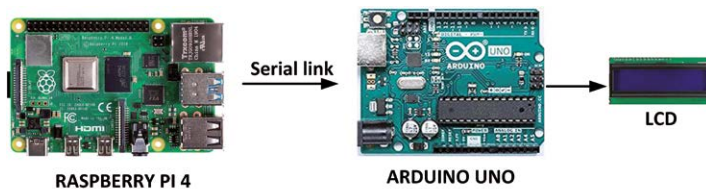


Figure 11.27 Block diagram of the project

Circuit diagram: Figure 11.28 shows the circuit diagram of the project. The serial output pin of the Raspberry Pi is connected to pin 3 of the Arduino Uno, where this pin is configured as serial input. A standard HD44780 compatible parallel LCD is connected to the Arduino Uno as shown in the figure. The output HIGH voltage of a Raspberry Pi pin is maximum +3.3V and this is not enough to drive the input of the Arduino Uno. A logic level converter module (Figure 11.29) is used to increase the output voltage level of Raspberry Pi to +5V. As shown in Figure 11.30, the one used in this project is a 2-channel module. A MOSFET transistor is used to convert +3.3V to +5V, and a resistive potential divider circuit is used to convert from +5V to +3.3V.

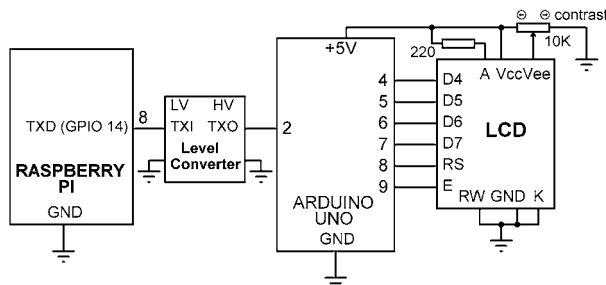


Figure 11.28 Circuit diagram of the project

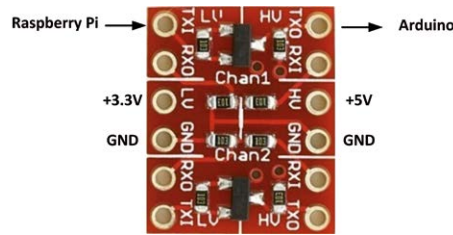


Figure 11.29 Logic level converter module

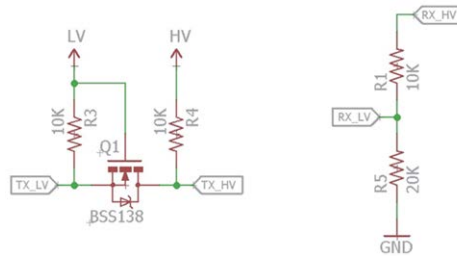


Figure 11.30 Circuit diagram of the logic level converter

Arduino Uno program: Figure 11.31 shows the Arduino Uno program (program: arduinolcd). At the beginning of the program, the software library and the LCD libraries are included in the program. The interface between the LCD and the Arduino Uno is defined. Here, the pins should be defined as follows: RS, E, D4, D5, D6, D7. The software library pins are defined such that pin 2 is input and pin 3 is output (not used in this project). Inside the setup routine, the serial line Baud rate is set to 9600, the LCD is initialized to 16 characters by 2 lines. The text Arduino is then displayed on the LCD.

The remainder of the program runs inside the main program loop. Here, the program checks if there is data at the serial port, and if so, function `readStringUntil()` is used to read a string of data until the terminating character '#' is detected (this character is sent by Raspberry Pi to indicate the end of its data) and the received data is stored in string variable `s1` in the form: `nn.n,mm#`. Then functions `indexOf()` and `substring()` is used to extract the temperature and humidity from string `s1`. The temperature is displayed at the top row as `T=nn.nC` and the humidity is displayed at the bottom row as `H=mm%`.

```

/*-----
 * This program receives the temperature and humidity data
 * from the Raspberry Pi over the serial line at 9600 Baud.
 * The temperature and humidity are displayed on an LCD
 *
 * File: Arduinolcd
 *-----*/
#include <SoftwareSerial.h>           // Serial library
#include <LiquidCrystal.h>           // LCD library
LiquidCrystal lcd(8,9,4,5,6,7);      // LCD connections
SoftwareSerial MySerial(2, 3);       // RX, TX

```

```

void setup()
{
    MySerial.begin(9600);           // Start serial
    lcd.begin(16,2);                // Init LCD
    lcd.print("Arduino");           // Display Arduino
    delay(1000);                    // Wait 1 second
}

void loop()
{
    if(MySerial.available() > 0)    // Data available?
    {
        lcd.clear();               // Clear LCD
        String s1 = MySerial.readStringUntil('#');
        int comma = s1.indexOf(','); // Position of comma
        String Temperature = s1.substring(0, comma);
        String Humidity = s1.substring(comma+1);
        lcd.setCursor(0,0);         // Cursor at 0,0
        lcd.print("T=");            // Display T=
        lcd.print(Temperature);     // Display Temp
        lcd.print("C");             // Display C
        lcd.setCursor(0,1);        // Cursor at 0,1
        lcd.print("H=");            // Display H=
        lcd.print(Humidity);        // Display humidity
        lcd.print("%");             // Display %
    }
}
    
```

Figure 11.31 Arduino Uno program

Node-RED flow program: Figure 11.32 shows the flow program of this project. In this project there are 4 nodes: an **inject** node to start the flow, an **openweathermap** node to get the local weather report, a **function** node to extract the temperature and humidity from this report, and a **serial out** node to send data to the Arduino Uno over the serial line.

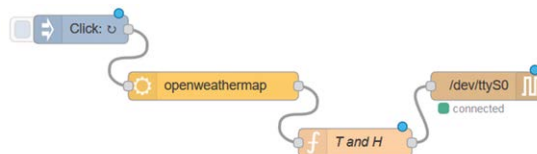


Figure 11.32 Flow program of the project

The steps are as follows:

- Create an **injection** node to repeat every 5 seconds
- Create an openweathermap node as before and enter the API key
- Create a **function** node named **T and H**, and enter the following statements inside this node. This node extracts the temperature and humidity, separates them with a comma and inserts the terminator character '#', click **Done**:

```
var T=msg.payload.tempc;
var H = msg.payload.humidity;
All = T + "," + H + "#";
msg.payload = All;
return msg;
```

- Create a **serial out** node and configure it as shown in Figure 11.33, click **Update** and then **Done**

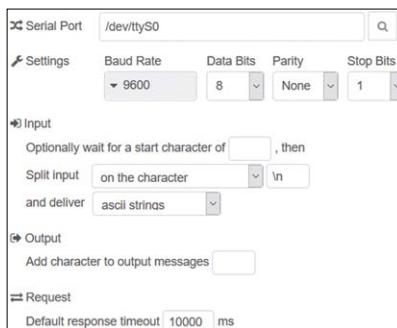


Figure 11.33 Configure node serial out

- Join the nodes and click **Deploy**.
- Build your circuit as shown in Figure 11.28, turn ON the Arduino Uno. You should see the temperature and humidity displayed on the LCD as shown in Figure 11.34



Figure 11.34 Displaying the temperature and humidity

Don't forget, after you finish working with your serial port, you may want to give this port back to the console. This is easily done by using the `raspi-config` utility by selecting option **5** (Interfacing Options) and then **P6** (Serial) and select **Yes**. Exit `raspi-config` and restart your Raspberry Pi.

11.7 Project 58 – Connecting Raspberry Pi and Arduino Uno using USB ports

Description: In the previous project we had to connect the serial output port of our Raspberry Pi to the serial input of Arduino Uno. In this project, we just connect the USB port of the Arduino to one of the USB ports of the Raspberry Pi. As in the previous project we get and display the local temperature and humidity on the Android Uno.

Aim: The aim of this project is to show how serial communication can be established using one of the USB ports of Raspberry Pi.

Block diagram: The block diagram of the project is shown in Figure 11.35. The Arduino is connected to the Raspberry Pi using a USB cable. There is no other connection between the two computers.

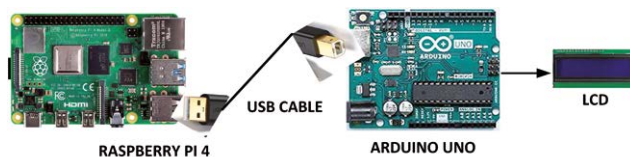


Figure 11.35 Block diagram of the project

Raspberry Pi USB port device name: Before using the Raspberry Pi USB ports, we have to know their device names. Perhaps the easiest way is to display the serial device list without the Arduino, and then connect Arduino to Raspberry Pi and display the device list again. This way, the device name of the Raspberry Pi can be found very easily. The command to display the serial devices is:

```
pi@raspberrypi:~ $ ls /dev/tty*
```

Figure 11.36 shows the serial devices on the author's Raspberry Pi before Arduino Uno was connected.

```
pi@raspberrypi:~ $ ls /dev/tty*
/dev/tty0  /dev/tty19  /dev/tty3  /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty1  /dev/tty2  /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty10 /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty11 /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty12 /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty13 /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyAMA0
/dev/tty14 /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyprintk
/dev/tty15 /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58  /dev/ttyS0
/dev/tty16 /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59
/dev/tty17 /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6
/dev/tty18 /dev/tty28  /dev/tty39  /dev/tty5  /dev/tty60
/dev/tty29 /dev/tty4  /dev/tty50  /dev/tty61
```

Figure 11.36 Raspberry Pi serial devices

The device list after connecting the Arduino Uno to the Raspberry Pi with a USB cable is shown in Figure 11.37.

```

pi@raspberrypi:~$ ls /dev/tty*
/dev/tty  /dev/tty19  /dev/tty30  /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0  /dev/tty2   /dev/tty31  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1  /dev/tty20  /dev/tty32  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10 /dev/tty21  /dev/tty33  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11 /dev/tty22  /dev/tty34  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12 /dev/tty23  /dev/tty35  /dev/tty45  /dev/tty56  /dev/ttyACM0
/dev/tty13 /dev/tty24  /dev/tty36  /dev/tty46  /dev/tty57  /dev/ttyAMA0
/dev/tty14 /dev/tty25  /dev/tty37  /dev/tty47  /dev/tty58  /dev/ttyprintk
/dev/tty15 /dev/tty26  /dev/tty38  /dev/tty48  /dev/tty59  /dev/ttyS0
/dev/tty16 /dev/tty27  /dev/tty39  /dev/tty49  /dev/tty6   /dev/ttyS1
/dev/tty17 /dev/tty28  /dev/tty4   /dev/tty50  /dev/tty60  /dev/ttyS2
/dev/tty18 /dev/tty29  /dev/tty40  /dev/tty51  /dev/tty61

```

Figure 11.37 Serial devices after Arduino Uno is connected

Comparing the two figures, it is obvious that the USB port device name is: **/dev/ttyACM0**. Now that we know the device name, we can modify our Arduino and Node-RED flow programs accordingly.

Arduino Uno program: The Arduino Uno USB port is connected to hardware serial port pins 0 (RX) and 1 (TX) and is named just **Serial**. Therefore, the only changes required in Figure 11.31 is to remove the software serial ports and replace **MySerial** with **Serial**. The modified program (program: **arduinousb**) listing is shown in Figure 11.38.

```

/*-----
 * This program receives the temperature and humidity data
 * from the Raspberry Pi over a USB link at 9600 Baud.
 * The temperature and humidity are displayed on an LCD
 *
 * File: Arduinousb
 *-----*/
#include <LiquidCrystal.h>           // LCD library
LiquidCrystal lcd(8,9,4,5,6,7);     // LCD connections

void setup()
{
  Serial.begin(9600);               // Start serial
  lcd.begin(16,2);                  // Init LCD
  lcd.print("Arduino");              // Display Arduino
  delay(1000);                      // Wait 1 second
}

void loop()
{
  if(Serial.available() > 0)        // Data available?
  {
    lcd.clear();                    // Clear LCD
    String s1 = Serial.readStringUntil('#');
    int comma = s1.indexOf(',');     // Position of comma
    String Temperature = s1.substring(0, comma);
    String Humidity = s1.substring(comma+1);
    lcd.setCursor(0,0);              // Cursor at 0,0

```

```

        lcd.print("T=");                                // Display T=
        lcd.print(Temperature);                          // Display Temp
        lcd.print("C");                                  // Display C
        lcd.setCursor(0,1);                              // Cursor at 0,1
        lcd.print("H=");                                // Display H=
        lcd.print(Humidity);                             // Display humidity
        lcd.print("%");                                  // Display %
    }
}

```

Figure 11.38 Modified Arduino Uno program

Node-RED Flow program: The only change required in the flow program in Figure 11.32 is to change the serial port name from **/dev/ttyS0** to **/dev/ttyACM0** in node **serial out**, everything else remains the same. That's all. Click **Deploy** and you should see the temperature and humidity displayed on the LCD as in Figure 11.34.

11.8 Project 59 – Sending serial data from Raspberry Pi to Arduino over RF Radio

Description: In the previous projects we sent data from the Raspberry Pi using a direct connection to the Arduino, either by connecting the Raspberry Pi transmit pin to Arduino receive pin, or by connecting the two devices together using a USB cable.

In this project, we use a pair of RF radio modules to send serial data from the Raspberry Pi to the Arduino Uno. As in the previous project, the temperature and humidity data are sent to the Arduino Uno which displays them on the LCD.

Aim: The aim of this project is to show how RF radio modules can be used to send (or receive) serial data from a Raspberry Pi.

Background information: RF radio communication modules are used in projects where there are no other means of establishing communication between two devices. e.g. there is no Wi-Fi, Internet, or Bluetooth connectivity. In this project the medium-range tRF Click module (see Figure 11.39) is used, one for Raspberry Pi, and one for Arduino Uno. tRF Click is a short-range RF communication module, operating at the 868MHz ISM license-free frequency band. This module is based on the LE70-868RF module from Telit. The module offers full AT command compatibility. An on-board UART is provided so that the host computer can send and receive messages via the UART. The module can operate with +3.3V to +5V, and it provides 500mW transmit power, and -117dBm receive sensitivity. The range of the tRF Click is quoted to be around 10 kilometers. An external antenna must be used with the module in order to achieve the maximum quoted range. Full details of the tRF module can be obtained from the link: <https://www.mikroe.com/trf-click>.



Figure 11.39 tRF Click module

tRF is a 2x8 pin module with the following pin names (pin 1 is at the top left corner, NC = No Connection):

1: STB	16: STS
2: RST	15: ACK
3: NC	14: TXD
4: NC	13: RXD
5: NC	12: NC
6: NC	11: NC
7: +3.3V	10: NC
8: GND	9: GND

Normally, only the following pins are used in most applications: +3.3V, GND, TXD, RXD

Block diagram: The block diagram of the project is shown in Figure 11.40. Two tRF Click modules are used to establish the RF radio communication between the devices.

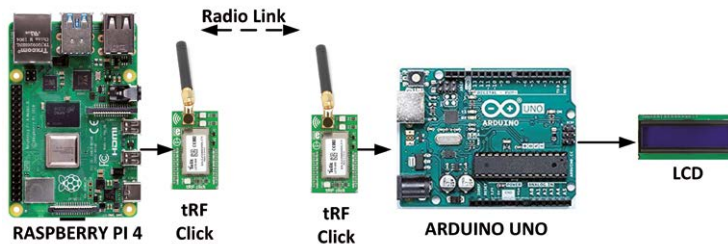


Figure 11.40 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 11.41. On the Raspberry Pi side, the serial output pin TXD (GPIO 14) is connected to the RXD input of the tRF module.

On the Arduino side, the TXD pin of tRF module is connected to serial input 2 of the Arduino through a logic level converter (see Figure 11.29) since the output HIGH level of tRF is only +3.3V (you may find that there may not be a need for a logic level converter here).

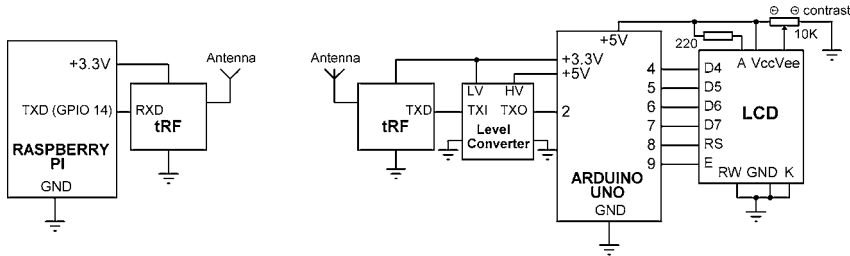


Figure 11.41 Circuit diagram of the project

Arduino Uno program: The Arduino Uno program is exactly the same as the one given in Figure 11.31 (program: `arduinolcd`), except that the Baud rate must be set to 19200 which is the default Baud rate of the tRF module.

Node-RED flow program: The Node-RED flow program for Raspberry Pi is exactly the same as the one given in Figure 11.32, where `/dev/ttyS0` is used as the serial port pin of the Raspberry Pi, and the Baud rate must be set to 19200.

Build the Arduino and Raspberry Pi circuits as shown in Figure 11.41. You should see the temperature and the humidity displayed on the LCD as shown in Figure 11.34.

11.9 Summary

In this chapter, we have learned how to use Node-RED **serial in** and **serial out** nodes with several projects.

In the next chapter, we will be learning how to use Raspberry Pi Sense HAT in projects with Node-RED.

Chapter 12 • Using Sense HAT

12.1 Overview

Sense HAT is an add-on board for Raspberry Pi containing a number of useful sensors and an LED array. HAT is an acronym for **H**ardware **A**ttached on **T**op. Sense HAT was an important component of the Astro Pi project, which was an educational Raspberry Pi sent to the International Space Station with the British astronaut Tim Peake to run code developed by children. The actual Astro Pi had some modifications and had metal casing to make it suitable for use in space.

Sense HAT includes sensors to measure temperature, humidity, pressure, accelerometer, gyroscope, and a magnetometer. In addition, an 8 x 8 independently programmable LED array is included on the board that can be programmed to display text and small images.

In this chapter, we will be designing various projects using the Node-RED Sense HAT node with the Sense HAT board. Before that, it is worthwhile to look at the features of the Sense HAT board. Detailed information on Sense HAT can be obtained from the link: magpi.cc/AstroPiGuide. Also from the **Essentials_SenseHAT_v1** (MagPi Essentials series), and from many other Internet sources.

12.2 The Sense HAT board

Figure 12.1 shows the Sense HAT board. We can identify the following components on the board:

- 8 x 8 LED array, having a 15-bit colour resolution
- Accelerometer, +/- 2/4/8/16 G
- Gyroscope, +/- 245/500/2000 DPS
- Magnetometer, +/- 4/8/12/16 Gauss
- Barometer, 260 – 1260 hPa absolute pressure
- Temperature sensor, +/-2°C accuracy, 0 - 65°C range
- Humidity sensor, +/- 4.5% accuracy, 20 – 80% range
- Graphics controller chip
- Five-button joystick with left, right, up, down, and enter movements

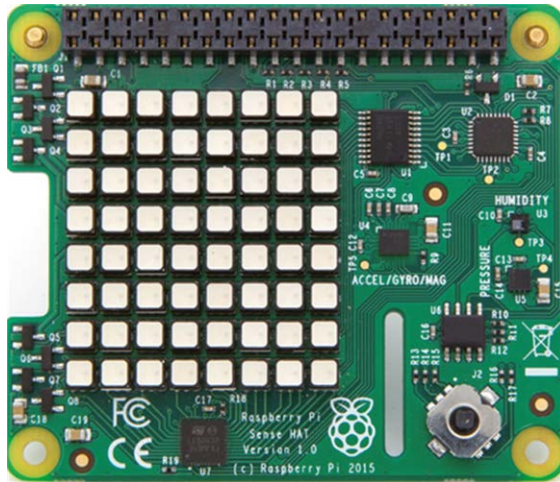


Figure 12.1 Sense HAT board

The LED matrix is very useful as it enables you to display text and also data from various sensors, to play games, etc. We will in later sections of this Chapter how to use display various items on the LED matrix.

A Node-RED simulator is available that can be downloaded from the Internet free of charge. The simulator allows you to create flows and interact with a virtual Sense HAT, without having the actual hardware. The simulator is useful to first try a project on the simulator before purchasing the hardware.

12.3 Node-RED Sense HAT nodes

Two Sense HAT nodes are available in Node-RED under the Raspberry Pi palette. As shown in Figure 12.2, these are the Sense HAT input node, and Sense HAT output node.



Figure 12.2 Sense HAT nodes

Sense Hat input node

Sense HAT input node reads the values of various sensors on the Sense HAT board and sends these readings. The sensors are grouped into three: **motion** events, **environment** events, and **joystick** events.

The motion events are the readings from the accelerometer, gyroscope, magnetometer, and compass heading, which are sent approximately 10 per second. The payload is the values of the sensors, and the topic is the motion. The sensor values are returned with the following units:

- Accelerometer (x,y,x) in Gs
- Gyroscope (x.y.z) in radians/s
- Orientation (roll, pitch, yaw) in degrees
- Compass heading in direction of North in degrees

The environmental events are readings from the following sensors: temperature, humidity, and pressure. The payload is the value returned by a sensor and the topic is the environment. The values are returned with the following units:

- The temperature in degrees Centigrade
- Humidity in percentage relative humidity
- Pressure in millibars

The joystick events are sent when the joystick is moved. The payload is the value returned and the topic is the joystick. The following values are returned:

- Key can be: UP, DOWN, LEFT, RIGHT, ENTER
- State of the key can be: 0 if the key has been released, 1 if the key has been pressed, and 2 if the key is being held down

For example, to get temperature, humidity, and pressure, we can create a function with 3 outputs, and enter the following statement inside the function:

```
var temperature = {payload: msg.payload.temperature};  
var humidity = {payload: msg.payload.humidity};  
var pressure = {payload: msg.payload.pressure};  
return [temperature, humidity, pressure];
```

Sense HAT output node

This node sends commands to the 8x8 LED matrix on the Sense HAT hardware. msg.payload is loaded with the commands to be sent. We will see in later sections how to use the output node in projects.

Example projects are given in the following sections to show how Sense HAT nodes can be used in Raspberry Pi projects. Make sure that the Sense HAT board is plugged on top of your Raspberry Pi board before using the projects.

12.4 Project 60 - Displaying the temperature, humidity, and pressure (environmental events)

Description: In this project, we will get temperature, humidity, and pressure from the Sense HAT board and display them in the form of gauges and charts.

Aim: The aim of this project is to show how the Environmental Events can be read and displayed.

Node-RED flow program: Figure 12.3 shows the flow program which consists just 8 nodes: a **Sense HAT** node to get the environmental variables, a **function** node to extract the temperature, humidity, and pressure, and 3 **gauge** nodes to display the temperature, humidity, and pressure, and 3 **chart** nodes to display the same variables.

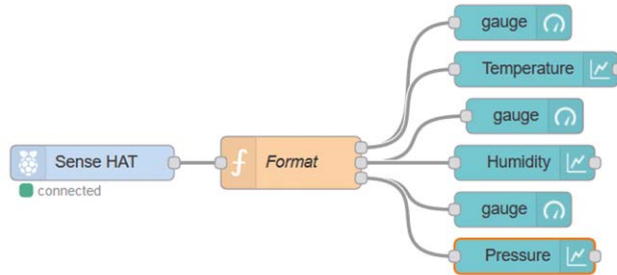


Figure 12.3 Flow program of the project

The steps are as follows:

- Create a **Sense HAT** node and set **Outputs** to Environmental Events
- Create function node, name it as Format and enter the following statements inside this node:

```
var temperature = {payload: msg.payload.temperature};
var humidity = {payload: msg.payload.humidity};
var pressure = {payload: msg.payload.pressure};
return [temperature, humidity, pressure];
```

- Create 3 gauge nodes and 3 chart nodes with the following configurations:

Gauge Node	Group	Label	Range	Units
Temperature	[Home]Ambient Temperature	gauge	0 to 35	C
Humidity	[Home]Ambient Humidity	gauge	0 to 100	%
Pressure	[Home]Ambient Pressure	gauge	900 to 1100	hPa

Chart Node	Group	Label	min-max
Temperature	[Home]Ambient Temperature	Temperature	0 to 35
Humidity	[Home]Ambient Humidity	Humidity	0 to 100
Pressure	[Home]Ambient Pressure	Pressure	900 to 1100

Join all nodes and click Deploy. You should see the temperature, humidity, and pressure displayed in the Dashboard. Notice that by default the gauges and charts are displayed vertically. We can, however, group the gauges and charts so that they are displayed horizontally, with the gauges and their corresponding charts displayed vertically together as shown in Figure 12.4. Grouping of the gauges and charts is shown in Figure 12.5.

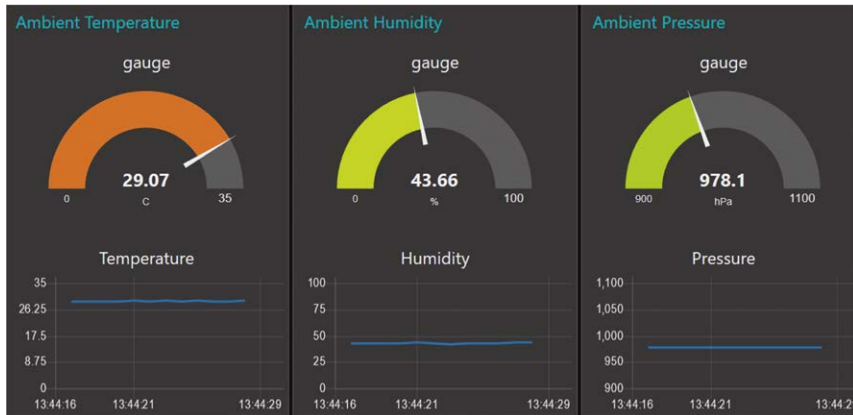


Figure 12.4 Displaying in Dashboard

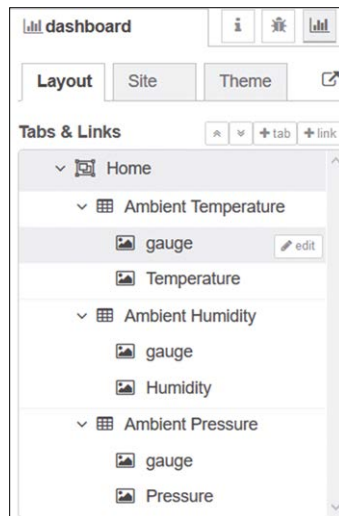


Figure 12.5 Grouping the gauges and charts

Notice that temperature is not displayed correctly. This is because the temperature sensor on the Sense HAT board is very close to the Raspberry Pi CPU and as a result of this the reading is not the correct ambient temperature reading. A correct reading can be obtained if the Sense HAT board is connected to Raspberry Pi using a ribbon cable so that the sensor is not close to the CPU.

12.5 Project 61 - Displaying the compass heading (motion events)

Description: In this project, we will display the compass heading continuously.

Aim: The aim of this project is to show how the compass heading can be displayed.

Node-RED flow program: Figure 12.6 shows the flow program which consists of just 3 nodes: a **Sense HAT** node to get the motion variables, a **function** node to extract the compass heading, and a Dashboard **text** node to display the heading dynamically.

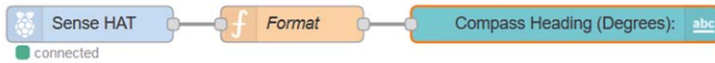


Figure 12.6 Flow program of the project

The steps are as follows:

- Create a **Sense HAT** node and set **Outputs** to Motion Events
- Create a **function** node, name it as **Format** and enter the following statements inside this node:

```
var CompassHead = {payload: msg.payload.compass};
return [CompassHead];
```

- Create a Dashboard **text** node, name it as **Compass Heading (Degrees):**, create a new group called **[Home]Sense HAT**
- Join the groups, click **Deploy**, and start your Dashboard. You should see the compass heading from North displayed dynamically as shown in Figure 12.7

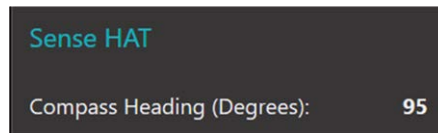


Figure 12.7 Displaying the compass heading

12.6 Project 62 - Displaying the acceleration (motion events)

Description: The acceleration (amount of G force) can be obtained in three dimensions of x, y, and z. In this project, we will display the acceleration continuously in all three dimensions.

Aim: The aim of this project is to show how acceleration can be displayed.

Node-RED flow program: Figure 12.8 shows the flow program which consists of 5 nodes: a **Sense HAT** node to get the motion variables, a **function** node with 3 outputs to extract the acceleration in 3 dimensions, and 3 Dashboard **text** nodes to display the acceleration in each direction.

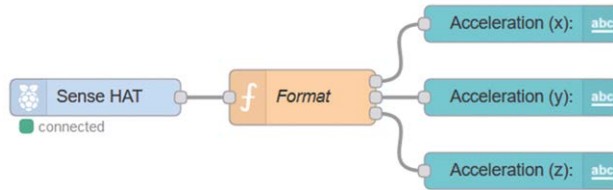


Figure 12.8 Flow program of the project

- Create a **Sense HAT** node and set **Outputs** to Motion Events
- Create a **function** node with 3 outputs, name it as **Format** and enter the following statements inside this node. This function returns the acceleration in 3 dimensions:


```

var accx = {payload: msg.payload.acceleration.x};
var accy = {payload: msg.payload.acceleration.y};
var accz = {payload: msg.payload.acceleration.z};
return [accx, accy, accz];
      
```
- Create 3 Dashboard **text** nodes, name them as **Acceleration (x):**, **Acceleration(y):**, and **Acceleration (z):**, use previously created Group **[Home]Sense HAT**
- Join the nodes and click **Deploy**. You should see the acceleration displayed in your Dashboard as shown in Figure 12.9

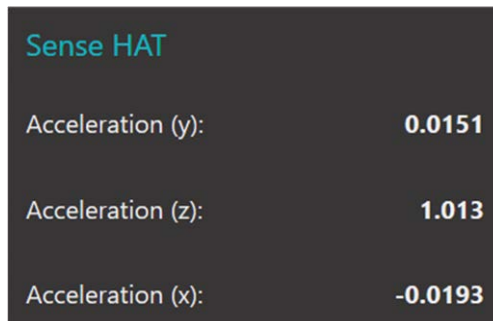


Figure 12.9 Displaying the acceleration in 3 dimensions

12.7 Project 63 - Displaying the orientation (motion events)

Description: In this project, we will display pitch, roll, and yaw dynamically.

Aim: The aim of this project is to show how the orientation can be displayed dynamically.

Background information: It is important to understand the difference between pitch, roll, and yaw. Perhaps the easiest way to understand these terms is to look at the movements of an aeroplane. Figure 12.10 shows an aeroplane with the terms pitch, roll, and

yaw explained graphically.

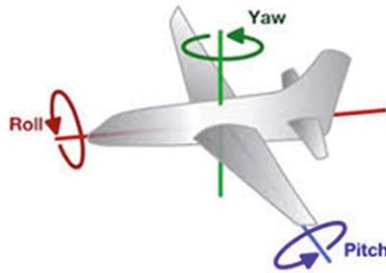


Figure 12.10 Pitch, roll, and yaw of an aeroplane

In Figure 12.11, the terms pitch, roll, and yaw as applied to Sense HAT are shown.

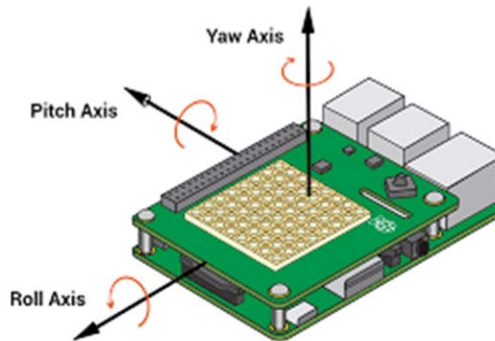


Figure 12.11 Pitch, roll, and yaw of Sense HAT

Node-RED flow program: The flow program is exactly the same as Figure 12.8, but the contents of **function** node and the 3 **text** nodes are modified. Figure 12.12 shows the flow program.

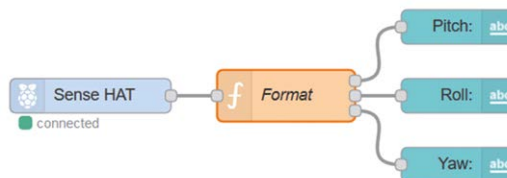


Figure 12.12 Flow program of the project

- Enter the following statements inside the **function** node:

```
var accx = {payload: msg.payload.orientation.pitch};
var accy = {payload: msg.payload.orientation.roll};
var accz = {payload: msg.payload.orientation.yaw};
return [accx, accy, accz];
```

- Change the **Labels** of the **text** nodes as shown in Figure 12.12. Figure 12.13 shows the pitch, roll, and yaw displayed in the Dashboard.

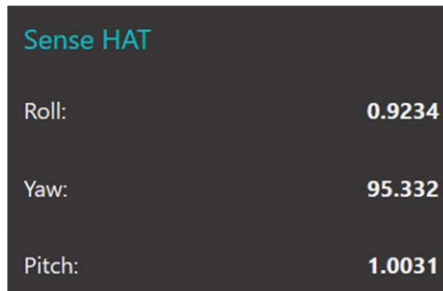


Figure 12.13 Displaying the pitch, roll, and yaw

12.8 Using the joystick

The joystick is in the Joystick Events group of Sense HAT. As described earlier in section 12.3, Joystick Events are sent when the joystick is moved. The payload is the value returned and the topic is the joystick. The returned values are UP, DOWN, RIGHT, LEFT, ENTER. The states of the keys are:

- 0: key has been released
- 1: key has been pressed
- 2: key is being held down

Perhaps the easiest way to understand the operation of the joystick is to set the **Outputs** of the Sense HAT node to Joystick events and connect a **debug** node to it (Figure 12.14). By moving or pressing the joystick the Debug window will show the action performed.

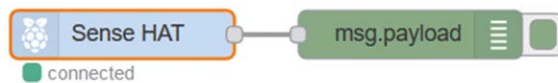


Figure 12.14 Flow program to test the joystick

For example, moving the joystick left and holding it down generates the message shown in Figure 12.15.

```

    ▶ { key: "LEFT", state: 2 }
    20/12/2019, 15:08:50 node: f6527996.096db8
    joystick : msg.payload : Object
    ▶ { key: "LEFT", state: 2 }
    20/12/2019, 15:08:50 node: f6527996.096db8
    joystick : msg.payload : Object
    ▶ { key: "LEFT", state: 2 }
    20/12/2019, 15:08:50 node: f6527996.096db8
    joystick : msg.payload : Object
    ▶ { key: "LEFT", state: 2 }
  
```

Figure 12.15 Moving the joystick left and holding it

12.9 Using the LED matrix

The **Sense HAT output** node is used to control the LED matrix on Sense HAT hardware. The commands to control the LED matrix are sent to msg.payload (multiple commands can be sent in a single message by separating them with a newline character).

Most of the LED matrix control statements are based on the LED coordinates (x, y) on the Sense HAT board. Figure 12.16 shows the LED co-ordinates in relation to the board.

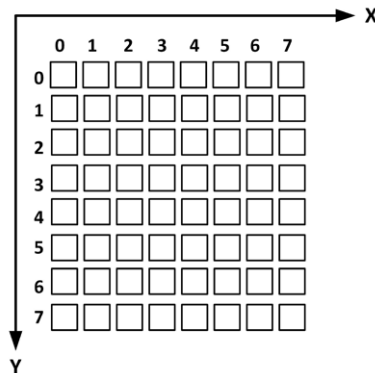


Figure 12.16 LED co-ordinates

Perhaps the easiest way to understand how the LED matrix operates is to create an LED matrix test flow program by connecting an **inject** node and a **function** node to the **Sense HAT output** node as shown in Figure 12.17, and then examining the LED matrix by writing statements inside **function** node.

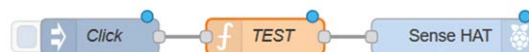


Figure 12.17 LED matrix test flow program

LED colours

The colour of an individual LED can be set using the statement:

```
"x,y,colour"
```

Where x and y must either be from 0 to 7 or a * to indicate the entire row or column. The colour can be one of the following:

HEX colour name (e.g. #aa991e)

RGB triple (e.g. 255, 180, 0)

HTML colour name

off

Example 1

Set the LED at co-ordinate 0,0 (i.e. at the top left corner when the USB sockets of Raspberry PI are facing the right-hand side) to red and turn ON the LED as shown in Figure 12.18.

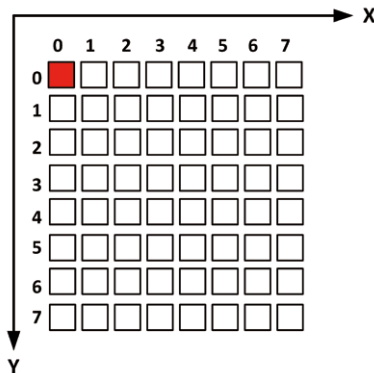


Figure 12.18 LED pattern for Example 1

Solution 1

Enter the following statement into the function:

```
msg.payload = "0,0,red";
return msg;
```

Example 2

Set all LEDs to a blue colour.

Solution 2

Enter the following statement into the function:

```
msg.payload = "*,*,blue";
return msg;
```


Example 3

Set LEDs at four corners to blue, red, green, and yellow as shown in Figure 12.19

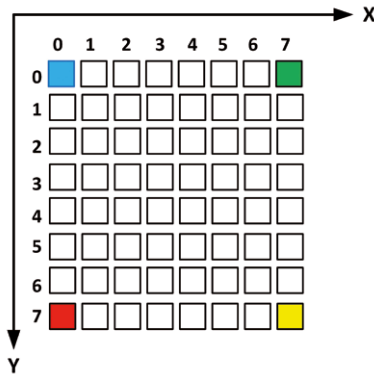


Figure 12.19 LED pattern for Example 3

Solution 3

Enter the following statement into the function:

```
msg.payload = "0,0,blue,0,7,red,7,0,green,7,7,yellow"
return msg;
```

Example 4

Set LEDs in a 4-pixel wide column to blue as shown in Figure 12.20.

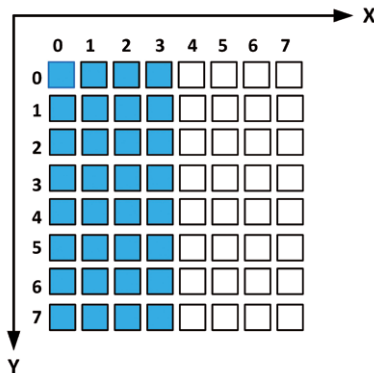


Figure 12.20 LED pattern for Example 4

```
msg.payload = "0-3,*,blue";
return msg;
```

Example 5

Set all LEDs in the second row to blue as shown in Figure 12.21.

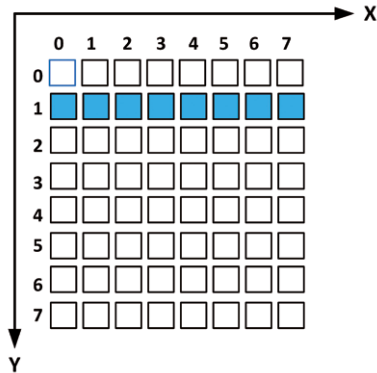


Figure 12.21 LED pattern for Example 5

Solution 5

Enter the following statement into the function:

```
msg.payload = "*,1,blue";
return msg;
```

Example 6

Set the diagonal LEDs to red as shown in Figure 12.22.

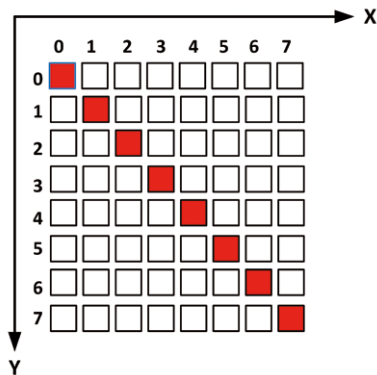


Figure 12.22 LED pattern for Example 6

Solution 6

Enter the following statement into the function:

```
msg.payload = "0,0,red,1,1,red,2,2,red,3,3,red,4,4,red,5,5,red,6,6,red,7,7,
red";
return msg;
```

We could have also written the following function to set the diagonal LEDs to red:

```
var L="";
for(i=0; i <= 7; i++)
{
    if(i != 7)
        L = L + String(i)+",""+String(i)+"red,";
    else
        L = L + String(i)+",""+String(i)+"red";
}
msg.payload = String(L);
return msg;
```

12.10 Project 64 – Random flashing LED lights having random colours

Description: In this project, we will flash LEDs on the LED matrix randomly. The colours are also random.

Aim: The aim of this project is to show how random numbers can be used to flash LEDs randomly, having random colours.

Node-RED flow program: The flow program is shown in Figure 12.23, and it consists of 11 nodes: an **inject** node that repeats every second, 5 **random** number nodes. Two of these nodes generate numbers between 0 and 7 that correspond to the LED co-ordinates, the other 3 nodes generate numbers from 0 to 255 to correspond to the RGB colours, a **join** node that joins all the random numbers and passes them to a **function** node. The **function** node outputs strings to flash the LEDs randomly having random colours. A **delay** node is also used to turn OFF all the LEDs every second so that only one LED is ON at any time.

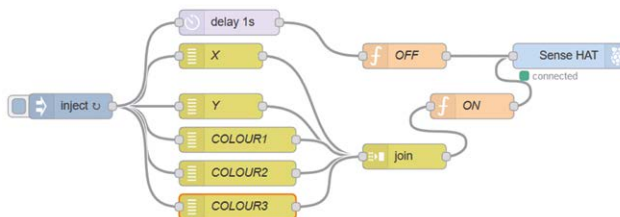


Figure 12.23 Flow program of the project

The steps are as follows:

- Create an **inject** node to repeat every second
- Create two **random** nodes named **X** and **Y** and configure them to generate random numbers between 0 and 7.
- Create three more **random** numbers named **COLOUR1**, **COLOUR2** and **COLOUR3** and configure them to generate random numbers between 0 and 255

- Create a **join** node and configure it as shown in Figure 12.24

Figure 12.24 Configure node join

- Create a **function** node named **ON** and enter the following statements inside this node:

```
var x = msg.payload[0];
var y = msg.payload[1];
var c1 = msg.payload[2];
var c2 = msg.payload[3];
var c3 = msg.payload[4];
msg.payload = x + "," + y + "," + c1 + "," + c2 + "," + c3;
return msg;
```

- Create another **function** node names **OFF** and enter the following statements:

```
msg.payload = "*,*,off";
return msg;
```

- Create a **Sense HAT** output node and join all the nodes. You should see LEDs turning **ON** and **OFF** randomly in different colours.

12.11 Project 65 – Display of temperature by LED count

Description: We have seen earlier in this Chapter how to control LEDs on the LED matrix. In this project, we will be displaying the temperature in real-time by turning the correct number of LEDs ON.

There are 64 LEDs in the LED matrix. For simplicity, we will assign each LED to 0.5°C so that the temperature can be displayed from 0°C (corresponding to no LEDs ON) to up to 32°C (corresponding to all the 64 LEDs ON). Therefore, by counting the number of LEDs that are ON and dividing the number by 2 we find the temperature.

Aim: The aim of this project is to show how the Sense HAT board can be used to read the ambient temperature and to display the temperature by turning ON the correct number of LEDs.

Node-RED flow program: The flow program is shown in Figure 12.25, and it consists of 4 nodes: a **Sense HAT input** node to read the ambient temperature, a **function** node with two outputs to format the readings and also to turn OFF the screen so that it can be updated, a delay node to limit the messages, and a **Sense HAT output** node to turn the appropriate number of LEDs ON.



```

> { key: "LEFT", state: 2 }
20/12/2019, 15:08:50 node: f6527996.096db8
joystick : msg.payload : Object
> { key: "LEFT", state: 2 }
20/12/2019, 15:08:50 node: f6527996.096db8
joystick : msg.payload : Object
> { key: "LEFT", state: 2 }
20/12/2019, 15:08:50 node: f6527996.096db8
joystick : msg.payload : Object
> { key: "LEFT", state: 2 }

```

Figure 12.25 Flow program of the project

The steps are as follows:

- Create a **Sense input** node and set **Outputs** to Environment events
- Create a **function** node named **Format** with two outputs, and enter the following statements inside this node. Variable **T** extracts the temperature from Sense HAT. Variable **OnCount** calculates the number of LEDs that must be ON. For example, if the temperature is 22°C then 44 LEDs must be turned ON. The remainder of the code is executed if the temperature is below or equal to 32°C. Variable **rows** is the number of rows of LEDs that must be ON. For example, if the temperature is 22°C, then **rows** is equal to 5 so rows: 0, 1, 2, 3, 4 should be ON. **cols** is the remaining LEDs that should be ON. For example, if the temperature is 22°C then **cols** is equal to 4. i.e. 4 more LEDs must be turned ON. Three **for** loops are used in the function. The outer loop iterates from 0 to **rows**, the inner loop iterates 8 times so that all the LEDs in a given row are turned ON. The last loop turns ON the remaining LEDs. Variable **L** stores the pattern that will be output from the node.

The second output of the function sends text **off** to the display to clear the screen so that the display can be updated. When a new LED is turned ON or OFF, the ones already ON remain ON. For example, if the temperature is 22°C, then we have 44 LEDs ON. If the temperature goes down to 20°C, then the ON count does not go down to 40. Although a command is sent to turn ON only 40 LEDs, the ones already ON do not go OFF. Therefore, we clear the screen by turning all LEDs OFF and then update with the new data.

If the temperature is greater than 32°C, then the middle LED (at co-ordinate 4, 4) is turned red to indicate that the temperature is outside the limit:

```
var L="";
var T = msg.payload.temperature;
var OnCount = parseInt(2*T);
if(T <= 32)
{
    var rows = parseInt(OnCount / 8);
    var cols = OnCount-rows*8;

    for(i = 0; i < rows; i++)
    {
        for(j = 0; j <= 7; j++)
        {
            L = L + String(i)+","+String(j)+",red,";
        }
    }
    for(j=0; j < cols; j++)
    {
        L = L + String(rows)+","+String(j)+",red,";
    }

    var l = L.length;
    msg.payload=L.substr(0,l-1);
}
else
msg.payload = "4,4,red";
var f = {payload: "*",*,off"};
return [msg, f];
```

- Create a **delay** node with the **Action** set to Rate limit and configure it to send 1 message every 30 seconds. This node is connected to the second output of node **function** so that the screen is cleared by sending the text off.
- Create a **Sense HAT output** node and join all the node, click **Deploy**. You should see that the temperature displayed on the LED matrix with the number of LEDs turned ON. In the example shown in Figure 12.26, 51 LEDs are turned ON. Therefore, the temperature is $51 / 2 = 25.5^{\circ}\text{C}$. Note that, as described earlier, the temperature reading of Sense HAT is not accurate since the sensor is close to the Raspberry Pi CPU.

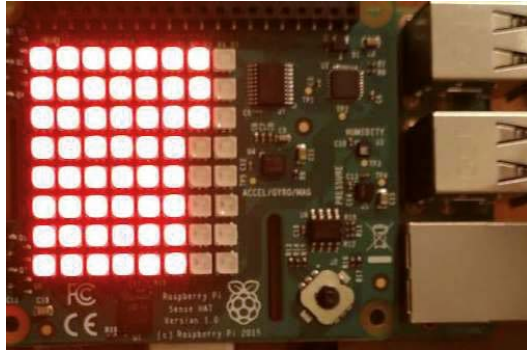


Figure 12.26 Displaying temperature 25.5°C

12.12 Displaying and scrolling data on LED Matrix

Data can easily be displayed and scrolled on the LED matrix by loading it to `msg.payload`. For example, to display and scroll text **Hello**, enter the following command into a function:

```
msg.payload = "Hello";  
return msg;
```

We can specify the background colour, text colour, and scroll speed by specifying: **background**, **color**, and **speed** respectively. Speed can take values 1 to 5, where 1 is slow and 5 is fast (default is 3). An example is shown below which specifies the background colour as yellow, text colour as red, and the speed as 2:

```
var txt = {payload: "Hello"};  
txt.background = "yellow";  
txt.color = "red";  
txt.speed = 2;  
return txt;
```

If the text is a single character, it will be displayed without scrolling. To scroll a single character, append a blank space after it, e.g. "X ".

12.13 Project 66 – Scrolling the pressure readings on the LED matrix

Description: In this project, we will scroll the pressure readings on the LED matrix. The background colour will be set to white, text colour to red, and the speed to 1.

Aim: The aim of this project is to show how the data can be scrolled on the LED matrix.

Node-RED flow program: The flow program is shown in Figure 12.27, and it consists of 3 nodes: a **Sense HAT input** node to read the ambient pressure, a **function** node to format the readings, a delay node to limit the rate of the messages to 1 every 5 seconds, and a **Sense HAT output** node to display the data on the LED matrix.

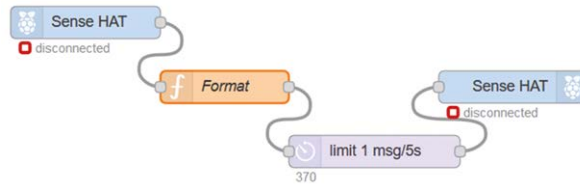


Figure 12.27 Flow program of the project

The steps are as follows:

- Create a **Sense input** node as in the previous project
- Create a **function** node and enter the following statements:

```
var T = {payload: msg.payload.pressure};
T.color = "red";
T.background = "white";
T.speed = 1;
return T;
```

- Create a **delay** node and set the **Action** to Rate limit to limit 1 message every 5 seconds. This node is required to slow down the messages coming from Sense HAT.
- Create a **Sense HAT output** node as before. Join all the nodes and click **Deploy**. You should see the ambient pressure displayed and scrolled on the LED matrix.

Some other useful Sense HAT commands are:

Rotate screen

To rotate the screen, enter command R followed by the angle. The angle must be 0, 90, 180, or 270. In the following example the screen is rotated 180 degrees:

```
var T = "Hello";
msg.payload = "R90\n" + T;
return msg;
```

Flip the screen

To flip the screen enter command F followed by V (vertical) or H (horizontal).

Set the screen brightness

To set the screen brightness, enter the command D followed by the level. The level must be 0 (low) or 1 (high).

12.14 Summary

In this chapter, we have learned how to use Sense HAT input and output nodes in various projects.

In the next chapter, we will be learning how to use Node-RED with the Arduino and develop several projects with the Arduino.

Chapter 13 • Node-RED with Arduino Uno

13.1 Overview

Arduino Uno is one of the most commonly used microcontroller development boards currently. There are hundreds of projects available on the Internet using the Arduino Uno with sensors, actuators, and display devices.

This Chapter is about using Node-RED with the Arduino Uno. The hardware details of the Arduino Uno are not given in this chapter as it is assumed that the readers are familiar with the Arduino Uno and have developed programs using this development board. Interested readers can find many tutorials, application notes, datasheets, and projects for the Arduino Uno on the Internet. The chapter describes how to install and use Node-RED for the Arduino Uno. Several projects are described in the chapter using Node-RED.

13.2 Installing Node-RED For Arduino Uno

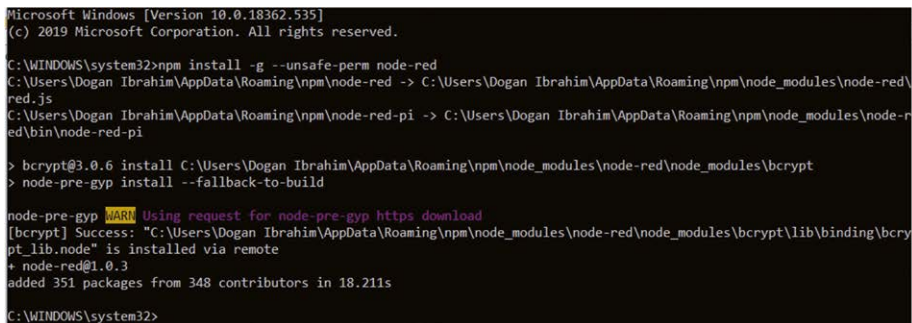
Before using Node-RED on our Arduino Uno, we have to install it on our Windows computer. The steps are as follows:

- Install **node.js** (64-bit or 32-bit according to your computer) from the following site:

<https://nodejs.org/en/download>

- Open the **Command** prompt on your computer as an administrator and enter the following command (see Figure 13.1):

`C:\Windows\system32> npm install -g --unsafe-perm node-red`



```
Microsoft Windows [Version 10.0.18362.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32> npm install -g --unsafe-perm node-red
C:\Users\Dogan Ibrahim\AppData\Roaming\npm\node-red -> C:\Users\Dogan Ibrahim\AppData\Roaming\npm\node_modules\node-red\node_modules\node-red.js
C:\Users\Dogan Ibrahim\AppData\Roaming\npm\node-red -> C:\Users\Dogan Ibrahim\AppData\Roaming\npm\node_modules\node-red\bin\node-red-pi

> bcrypt@3.0.6 install C:\Users\Dogan Ibrahim\AppData\Roaming\npm\node_modules\node-red\node_modules\bcrypt
> node-pre-gyp install --fallback-to-build

node-pre-gyp WARN Using request for node-pre-gyp https download
[bcrypt] Success: "C:\Users\Dogan Ibrahim\AppData\Roaming\npm\node_modules\node-red\node_modules\bcrypt\lib\binding\bcrypt_lib.node" is installed via remote
+ node-red@1.0.3
added 351 packages from 348 contributors in 18.211s

C:\WINDOWS\system32>
```

Figure 13.1 Install Node-RED

- To test the installation, enter the following command:

`C:\Windows\system32> node-red`

- You should see messages displayed similar to the ones shown in Figure 13.2. If you can see these messages, it means that Node-RED is installed correctly on your PC

```
C:\WINDOWS\system32\node-red
22 Dec 18:09:45 - [info]

Welcome to Node-RED
=====

22 Dec 18:09:45 - [info] Node-RED version: v1.0.3
22 Dec 18:09:45 - [info] Node.js version: v12.14.0
22 Dec 18:09:45 - [info] Windows_NT 10.0.18362 x64 LE
22 Dec 18:09:46 - [info] Loading palette nodes
22 Dec 18:09:46 - [info] Settings file : C:\Users\Dogan Ibrahim\.node-red\settings.js
22 Dec 18:09:46 - [info] Context store : 'default' [module-memory]
22 Dec 18:09:46 - [info] User directory : C:\Users\Dogan Ibrahim\.node-red
22 Dec 18:09:46 - [warn] Projects disabled : editorTheme.projects.enabled=false
22 Dec 18:09:46 - [info] Flows file : C:\Users\Dogan Ibrahim\.node-red\flows_DESKTOP-U7VQMOI.json
22 Dec 18:09:46 - [info] Creating new flow file
22 Dec 18:09:46 - [warn]
```

Figure 13.2 Testing your Node-RED installation

- Now start your web browser and enter the following address to start your Node-RED in your browser (do not close your command window):

<http://127.0.0.1:1880>

- You should now see the Node-RED startup screen as shown in Figure 13.3 (only part of the screen is shown here).

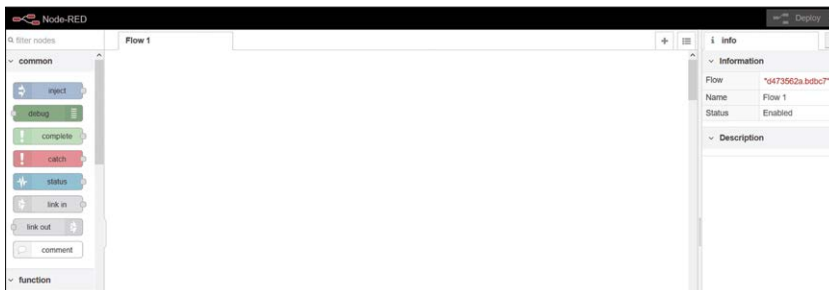


Figure 13.3 Node-RED startup screen

- Install the Arduino node. Click **Menu -> Manage palette** and click **Install**
- Enter **node-red-node-arduino** and click **install**
- You should see two nodes added to the Node Palette: **arduino in** and **arduino out**

The Arduino must be connected to the host computer via USB serial connection. We cannot use both the Arduino IDE and Arduino nodes at the same time as they will conflict. We will have to exit from the Arduino IDE in order to use Node-RED. Alternatively, we must stop Node-RED if we wish to program the Arduino using the IDE.

The **Firmata** protocol is used for communicating between an Arduino and the host computer, providing direct access to the GPIO pins. Before developing our programs, we have to load the Firmata into our Arduino Uno. The steps are as follows:

- Stop Node-RED
- Connect your Arduino to the PC via the USB cable
- Start Arduino IDE
- Select Arduino/Genuino Uno as the target board
- Make a note of the serial port name (e.g. COM5)
- Click Files -> Examples -> Firmata -> Standard Firmata
- Click Sketch -> Verify/Compile and then click Sketch -> Upload
- Exit Arduino IDE

You should now start Node-RED on your Windows, and then start the dashboard on your web browser:

```
C:\Windows\system32> node-red
```

<http://127.0.0.1:1880>

We are now ready to develop Arduino programs using Node-RED. A simple example program is given in the next section to show how Node-RED can be used with the Arduino Uno.

13.3 Project 67 – Flashing LED

Description: This is a very simple program where the LED on the Arduino Uno board at port 13 is flashed every second.

Aim: The aim of this project is to show how the on-board LED at port pin 13 can be flashed every second.

Node-RED flow program: Figure 13.4 shows the flow program which consists of just 3 nodes: an inject node that repeats every second, a function node that toggles its output, and an Arduino out node.



Figure 13.4 Flow program of the project

The steps are as follows:

- Create an **inject** node, name it as **1 second** and configure it to repeat every second
- Create a **function node**, name it as **Toggle LED** and insert the following statements inside this node. In this node, a context variable is toggled so that its value changes between 0 and 1. Remember that context variables keep their values inside the node:

```
context.led = !context.led || 0;
msg.payload = context.led;
return msg;
```

- Create an **arduino out** node and configure it as shown in Figure 13.5. Notice that the serial port number found earlier (COM5) is used in this node.

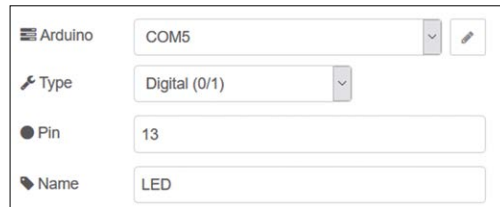


Figure 13.5 Configure node arduino out

- Join all the nodes as shown in Figure 13.4 and click **Deploy**.
- You should see the on-board LED flashing every second

13.4 Project 68 – Displaying the ambient temperature in the debug window

Description: In this project, the ambient temperature is measured using an analog temperature sensor chip.

Aim: The aim of this project is to show how **arduino in** node can be used to read analog input data.

Circuit diagram: An LM35DZ type analog temperature sensor chip is used in this project. The sensor is connected to +5V and its output is connected directly to analog input A0 of the Arduino Uno. Figure 13.6 shows the circuit diagram of the project.

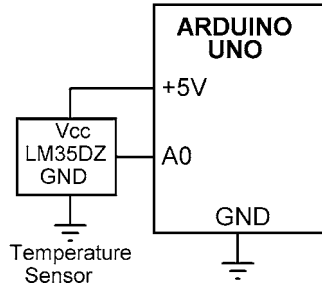


Figure 13.6 Circuit diagram of the project

Node-RED flow program: Figure 13.7 shows the flow program which consists of just 3 nodes: an **arduino in** node to receive the temperature readings, a **function** node to format the data, and a **debug** node to display the data in the Debug window.

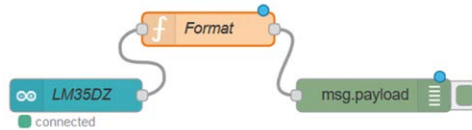


Figure 13.7 Flow program of the project

The steps are:

- Create an **arduino in** node and configure as shown in Figure 13.8

Figure 13.8 Configure node arduino in

- Create a **function** node and enter the following statements:

```
var T = msg.payload * 5000.0 / 1024.0;
var mv = T / 10.0;
msg.payload = "T = " + mv.toFixed(2) + "C";
return msg;
```

- Create a debug node, join the nodes, click Deploy.
- The temperature will be displayed in the Debug window as shown in Figure 13.9



Figure 13.9 Displaying the temperature

13.5 Project 69 – Displaying the ambient temperature in the dashboard

Description: In this project, the ambient temperature is measured as in the previous project and is displayed in node text of the Dashboard.

Aim: The aim of this project is to show how Dashboard nodes can be used in Arduino projects.

Circuit diagram: The circuit diagram of the project is as in Figure 13.6

Node-RED flow program: Figure 13.10 shows the flow program which consists of just 3 nodes: an **arduino in** node to receive the temperature readings, a **function** node to format the data, and a Dashboard text node.

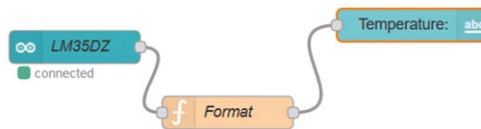


Figure 13.10 Flow program of the project

Before using a Dashboard node, we have to install the Dashboard Palettes. The steps are:

- Click **Menu -> Manage Palette** and click **Install**
- Enter **node-red-dashboard** and click **install**
- You should see the Dashboard nodes added to your Node Palette

The steps to develop the flow program are as follows:

- Create an **arduino in** node as in the previous project
- Create a **function** node as in the previous project

- Create a Dashboard **text** node and configure it as in Figure 13.11

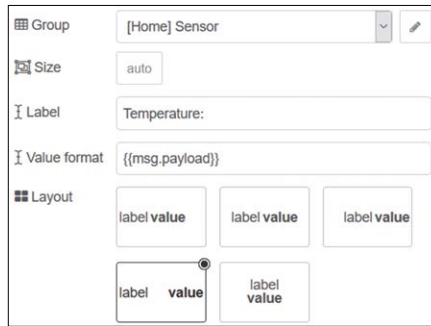


Figure 13.11 Configure the text node

- Join all nodes and click **Deploy**. Start the Dashboard by entering the following into your web browser. You should see the ambient temperature displayed as shown in Figure 13.12 (notice that the Style is set to Dark in this display):

127.0.0.1:1880/ui/

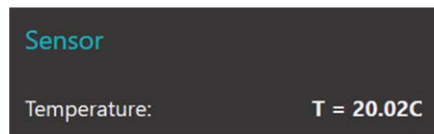


Figure 13.12 Displaying the temperature with node text

13.6 Project 70 – Displaying the ambient temperature as gauge and chart

Description: In this project, the ambient temperature is measured as in the previous project and is displayed in the form of a gauge and chart.

Aim: The aim of this project is to show how the Dashboard gauge and chart can be used in Arduino projects.

Circuit diagram: The circuit diagram of the project is as in Figure 13.6

Node-RED flow program: Figure 13.13 shows the flow program which consists of 4 nodes: an Arduino in node to read the temperature, a function node to format the temperature, a gauge and a chart nodes to display the temperature.



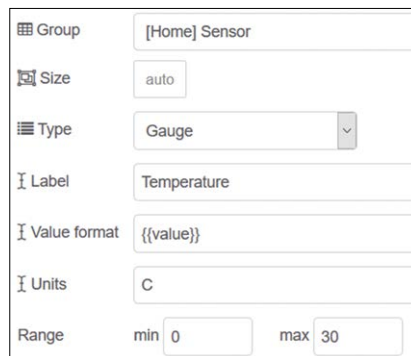
Figure 13.13 Flow program of the project

The steps are:

- Create an **arduino in** node as in the previous project
- Create a **function** node and enter the following statements:

```
var T = msg.payload * 5000.0 / 1024.0;
var mv = T / 10.0;
var Temp = mv.toFixed(2);
msg.payload = Temp;
return msg;
```

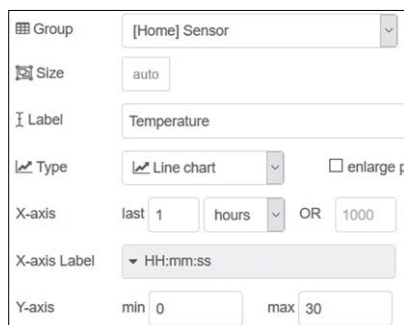
- Create a **gauge** node and configure as shown in Figure 13.14



The screenshot shows the configuration for a Gauge node. The 'Group' is '[Home] Sensor'. The 'Size' is set to 'auto'. The 'Type' is 'Gauge'. The 'Label' is 'Temperature'. The 'Value format' is '{{value}}'. The 'Units' are 'C'. The 'Range' is set with 'min' at 0 and 'max' at 30.

Figure 13.14 Configure the node gauge

- Create a **chart** node and configure as shown in Figure 13.15



The screenshot shows the configuration for a Chart node. The 'Group' is '[Home] Sensor'. The 'Size' is set to 'auto'. The 'Label' is 'Temperature'. The 'Type' is 'Line chart'. The 'X-axis' is set to 'last 1 hours' with an 'enlarge p' checkbox. The 'X-axis Label' is 'HH:mm:ss'. The 'Y-axis' is set with 'min' at 0 and 'max' at 30.

Figure 13.15 Configure the node chart

- Click **Deploy** and you should see the temperature displayed in the Dashboard using a gauge and a chart as shown in Figure 13.16

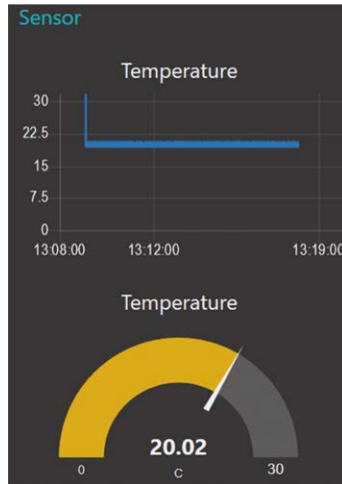


Figure 13.16 Displaying the temperature using gauge and chart

13.7 Using the Arduino Uno serial port

The readers will find that there is a limited number of nodes when using the Arduino with Node-RED. For example, at the time of writing this book, there was no LCD node that could be used with the Arduino. The solution to this problem is as follows:

We can connect the Arduino to one of the USB ports of Raspberry Pi and then write code using the Arduino IDE. The output can then be sent to Raspberry Pi via the serial port. The Raspberry Pi can receive the data using the serial node and process it as we have seen in earlier chapters of this book. This approach, of course, requires code to be developed for the Arduino.

An example project is given below using this idea.

13.7.1 Project 71 – Using the DHT11 with the Arduino

Description: In this project, a DHT11 temperature and humidity sensor are used and the ambient temperature and humidity data are displayed in the form of a gauge every 5 seconds.

Aim: The aim of this project is to show how the serial node can be used to receive data from the Arduino Uno.

Circuit diagram: The circuit diagram of the project is shown in Figure 13.17. The DHT11 is connected to digital port pin 3 of the Arduino Uno. The Arduino Uno is connected to one of the USB ports of Raspberry Pi using a USB cable. The temperature and humidity data are sent through this cable.

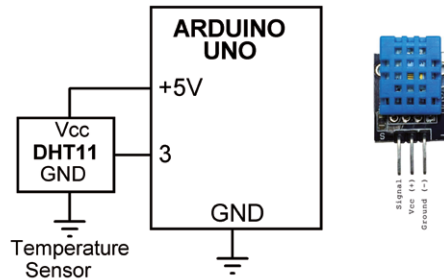


Figure 13.17 Circuit diagram of the project

Arduino Uno program: Before using the DHT11 with the Arduino Uno, we have to include the DHT11 library in our Arduino library. The steps are as follows:

- Start the Arduino IDE
- Click **Sketch -> Include Library -> Manage Libraries**
- Enter DHT11
- Click to install **DHT sensor library by Adafruit Version 1.3.8** (or a later version)
- Close the window
- Close and re-start the Arduino IDE

The Arduino program (program: **dht11arduino**) is shown in Figure 13.18. At the beginning of the program, pin 3 is assigned to DHT11 and DHT11 is defined as the sensor type (not DHT22). Inside the setup routine, the serial port is initialized to 9600 Baud and the DHT11 is initialized. The remainder of the program is executed inside the main program loop. Here, the humidity and temperature are read from the sensor and are then sent to the serial port. This process is repeated after 5 seconds.

```
#include <DHT.h>
#define DHTPin 3 // DHT11 pin
#define DHTType DHT11 // DHT11 type
DHT dht(DHTPin, DHTType); // Initialize

String Humidity, Temperature;
float Hum, Temp;

void setup()
{
    Serial.begin(9600); // Start serial
    dht.begin(); // Initialzie DHT11
```

```

}

void loop()
{
    Hum = dht.readHumidity();           // Get humidity
    Temp= dht.readTemperature();       // Get temperature
    Humidity = String(Hum);            // To string
    Temperature = String(Temp);        // To string
    Serial.print(Humidity);            // Send to serial port
    Serial.print(",");                 // Insert comma
    Serial.println(Temperature);        // Send to serial port
    delay(5000);                       //5 seconds delay
}

```

Figure 13.18 Arduino Uno program

Node-RED flow program: Re-start your Raspberry Pi, and then start Node-RED running on your Raspberry Pi. Start the Dashboard by entering (enter the IP address of your own Raspberry Pi) on your PC:

192.168.1.202:1880

Figure 13.19 shows the flow program which consists of 4 nodes: a serial in node to read the temperature and humidity data from the Arduino, a function node to format the received data, two gauges to display the temperature and the humidity.



Figure 13.19 Flow program of the project

The steps are:

- Create a **serial in** node with the configuration shown in Figure 13.20. Notice that the Serial Port name is set to **/dev/ttyACM0**, which is device name of the Raspberry Pi USB port.

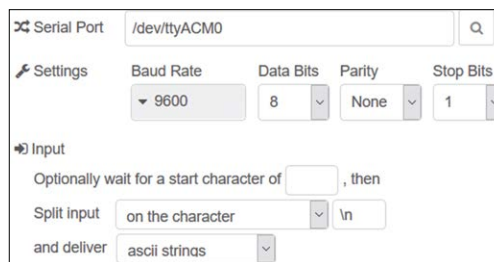


Figure 13.20 Configure node serial

- Create a **function** node with two outputs and enter the following statements (or, use the `split(",")` function to extract the humidity and the temperature:

```
var H = {payload: msg.payload.substr(0,5)};
var T = {payload: msg.payload.substr(6,5)};
return [H, T];
```

- Create two **gauge** nodes. For the humidity, node set the **Label** to Humidity, **min** and **max** values to 0 and 100, and the **Units** to % respectively. For the temperature, node set the **Label** to Temperature, **min** and **max** values to 0 and 30, and **Units** to C
- Join all the nodes, click **Deploy**, and start the Dashboard by entering your own IP address:

192.168.1.202:1880/ui/

- You should see the humidity and temperature displayed as shown in Figure 13.21

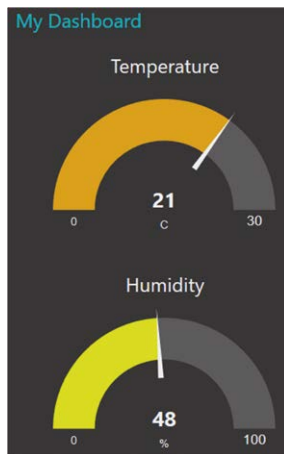


Figure 13.21 Displaying the humidity and temperature

13.8 Summary

In this chapter, we have learned how to use Node-RED with Arduino Uno, and several projects are given on this topic.

In the next chapter, we will look at how to use Node-RED with ESP32 processors.

Chapter 14 • Using the ESP32 DevkitC with Node-RED

14.1 Overview

ESP32 is one of the popular microcontrollers used currently by students, practicing electronics engineers, hobbyists, and many other people interested in microcontroller-based embedded system design. In this chapter, we will be learning how to use Node-RED with ESP32 microcontrollers. In this chapter, we will use the ESP32 DevKitC development board. It is assumed that the readers are familiar with the ESP32 processor and the ESP32 DevKitC development board and have used it in at least one project. In the project in this chapter, the ESP32 processor is programmed using the Arduino IDE. It is assumed that the Arduino IDE is loaded with the **ESP32 Dev Module** software. If this is not the case, then you should load this software to your Arduino IDE. The steps on installing the ESP32 support to the Arduino IDE on a Windows PC are given in this section. If you have already installed ESP32 on Arduino IDE then you must delete the **Espresif** folder from your Arduino directory before re-installing it.

The steps to install the ESP32 on Arduino IDE are as follows:

- Download and install the latest version of Arduino IDE from the following web site:

<https://www.arduino.cc/en/Main/Software>

- Open your Arduino IDE and click **File -> Preferences** to open the Preferences window. Locate text box **Additional Board Manager URLs** at the bottom of the window and enter the following text as shown in Figure 14.1. If the text box contains another URL, add the new URL after separating it with a comma:

https://dl.espressif.com/dl/package_esp32_index.json

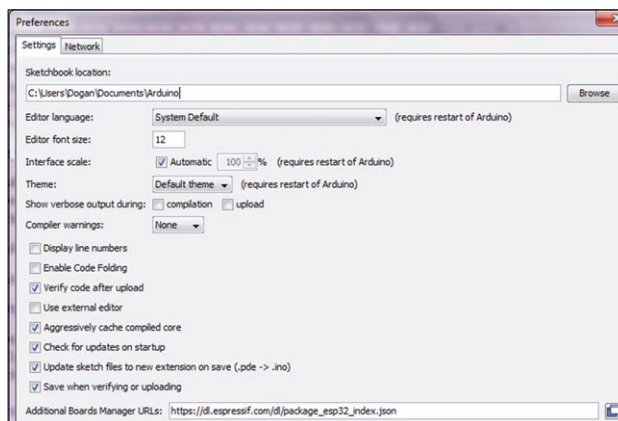


Figure 14.1 Preferences window

- Click **OK**

- Click **Tools -> Board -> Board Managers** window and search for ESP32 as shown in Figure 14.2

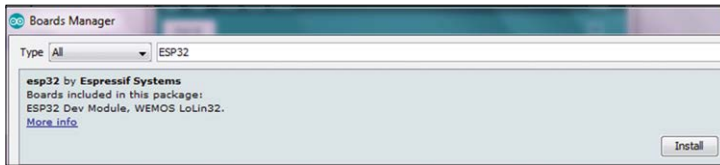


Figure 14.2 Search for ESP32

- Click the **Install** button. You should see the Installed message as shown in Figure 14.3. **Close** the window.

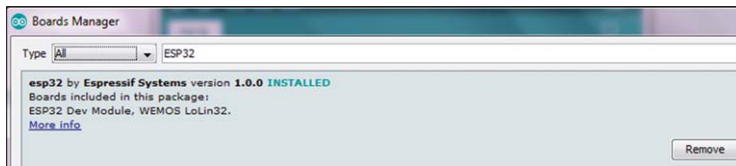


Figure 14.3 ESP32 installed

- If the software is installed correctly, then you should see the ESP32 Dev Module which is the board we will be using with our ESP32 DevKitC. Select **Tools -> Board -> ESP32 Dev Module** as shown in Figure 14.4.

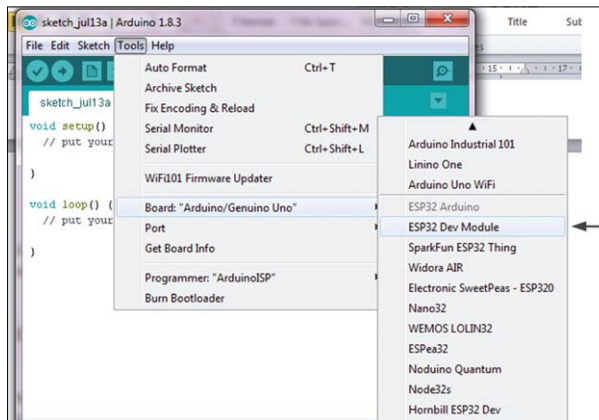


Figure 14.4 Select the ESP32 Dev Module

As a reminder, Figure 14.5 shows the pin configuration of the ESP32 DevKitC development board. Interested readers can get a large amount of data on the ESP32 processor from the Internet:

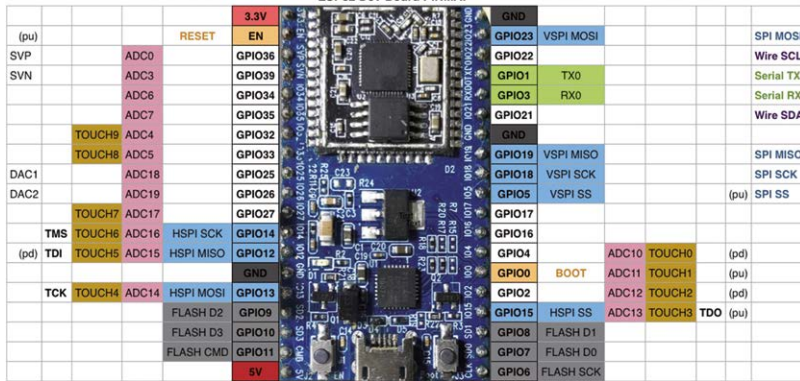


Figure 14.5 ESP32 DevKitC development board

14.2 ESP32 DevKitC and Node-RED

In this Chapter, we will be running Node-RED on a Raspberry Pi and then communicate with an ESP32 DevKitC I/O ports using a communications protocol. In practice, we can use TCP or UDP, or serial communication as shown in Figure 14.6. In the next project, we will be using serial communication.

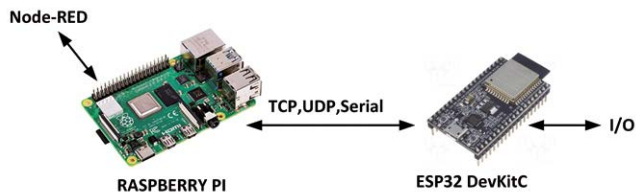


Figure 14.6 Raspberry Pi to ESP32 DevKitC interface

14.3 Project 72 – Controlling an LED connected to ESP32 DevKitC

Description: This is a very simple project. In this project, an LED is connected to one of the I/O ports of the ESP32 DevKitC through a current limiting resistor. The LED is turned ON or OFF by clicking inject buttons in Node-RED.

Aim: The aim of this project is to show how the ESP32 DevKitC input/output ports can be controlled from a Raspberry Pi which is programmed using Node-RED.

Block diagram: The block diagram of the project is shown in Figure 14.7.

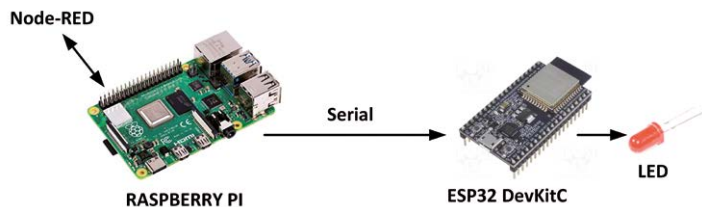


Figure 14.7 Block diagram of the project

Circuit diagram: Figure 14.8 shows the circuit diagram of the project. The LED is connected to GPIO 21 of the ESP32 DevKitC. Serial port output TXD (GPIO 14, physical pin 8) of the Raspberry Pi is connected to GPIO 23 of the ESP32 DevKitC. The ESP32 DevKitC has 3 UART pins as follows:

RXD (U0)	GPIO 3
TXD (U0)	GPIO 1
RXD (U1)	GPIO 9
TXD (U1)	GPIO 10
RXD (U2)	GPIO 16
TXD (U2)	GPIO 17

U0 is used by the USB to PC serial communication port. In some devices, U2 is reserved for the on-board flash and may not be available for serial communication. In general, most other pins can be configured for serial communication. In this project GPIO, 23 is used.

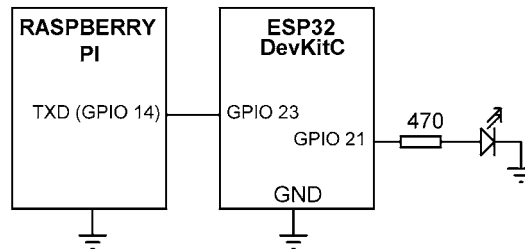


Figure 14.8 Circuit diagram of the project

ESP32 DevKitC Program: The program listing (program: **esp32nodered**) is shown in Figure 14.9, which is written using the Arduino IDE. At the beginning of the program, **HardwareSerial** library is included in the program, RXD and TXD (not used in this project) port pins are defined, and the LED is assigned to GPIO port 21. Inside the setup routine, the serial port is configured to 9600 Baud, 8 bits, no parity, and 1 stop bit. The LED is also configured as an output. The remainder of the program is executed in a loop. Inside this loop, the program waits to receive data from the Raspberry Pi. The received data is stored in a string variable called **buffer**. If the received data is **ON#** (where # is the terminator character) then the LED is turned ON. If on the other hand, the received data is **OFF#** then the LED is turned OFF. The program uses the serial port data receive function `readStringUntil()` which reads a string from the serial port until the specified character is encountered.

```

#include "HardwareSerial.h"
#define RXD 23
#define TXD 22

HardwareSerial Ser(1);
String buffer;
#define LED 21
  
```

```

void setup()
{
    pinMode(LED, OUTPUT);
    Ser.begin(9600, SERIAL_8N1, RXD, TXD);
}

void loop()
{
    while(Ser.available() > 0)
    {
        buffer = Ser.readStringUntil('#');
        if(buffer[0] == 'O' && buffer[1] == 'N')
            digitalWrite(LED, HIGH);
        else if(buffer[0] == 'O' && buffer[1] == 'F' && buffer[2] == 'F')
            digitalWrite(LED, LOW);
    }
}

```

Figure 14.9 ESP32 DevKitC program listing

Click **Sketch -> Compiler/Verify** to compile the program. After a successful compilation, you must press the **Boot** button on ESP32 DevKitC (the button at the bottom right-hand side) and keep it pressed and click **Sketch -> Upload**. When the uploading is complete, release the **Boot** button and press the **EN** button to reset and start the program running on your ESP32 DevKitC.

Node-RED flow program: Figure 14.10 shows the flow program which consists of just 3 nodes: two inject nodes to send the ON# or OFF# commands to the serial port, and a serial out port to send the commands to the ESP32 DevKitC. You should disable the console serial port (/dev/ttyS0) as described in Chapter 11 so that you can use the Raspberry Pi serial port.

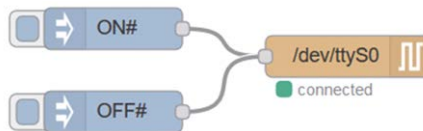


Figure 14.10 Flow program of the project

The steps are as follows:

- Create two **inject** nodes, set the Payload to ON# in one of them, and to OFF# in the other one.
- Create a **serial out** node and configure it as shown in Figure 14.11

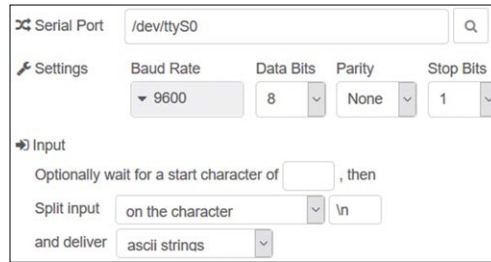


Figure 14.11 Configure node serial out

- Join all the nodes as in Figure 14.10 and click **Deploy**.
- Build the circuit as shown in Figure 14.8. Click the button of the **inject** node ON#, the LED should turn ON. Click the button of the **inject** node OFF#, the LED should turn OFF.

14.4 Summary

In this chapter, we have learned how to connect the ESP32 DevKitC to Raspberry Pi using the serial link at 9600 Baud. Node-RED was started on Raspberry Pi. By sending commands from Raspberry Pi to the ESP32 DevKitC we can easily control the LED. Interested readers can develop more complex ESP32 processor-based projects using the technique described in this chapter.

In the next chapter, we will be learning how to use Amazon Alexa in Node-RED based projects.

Chapter 15 • Using Amazon Alexa in Node-RED projects

15.1 Overview

Alexa is a sound-based virtual assistant device developed by Amazon in 2014 and first used in Amazon Echo and Amazon Echo Dot. Alexa interacts with the user through sound and it can obey the commands given by a speaker. For example, Alexa can play a piece of required music, it can provide weather reports, ask questions, give traffic reports, give the current news, give recipes, translate words to other languages, and many more. Alexa can also be used to control a device through sound commands and therefore it can be used in home automation. The device is activated by giving the wake-word Alexa. The user can then communicate with Alexa. Alexa is connected to the local Wi-Fi router of the user and it gets the answers to the user questions from the Amazon cloud. The latest model of Alexa at the time of writing this book was the 3rd generation devices. Figure 15.1 shows the 3rd Gen Alexa Dot device.



Figure 15.1 3rd Gen Alexa Dot

In this chapter, we will be using Node-RED with Alexa and Raspberry Pi and learn how we can control the GPIO ports of Raspberry Pi by giving spoken commands to Alexa.

15.2 Project 73 – Controlling an LED using Alexa

Description: In this project, an LED is connected to one of the Raspberry Pi GPIO ports. The LED is turned ON and OFF by giving spoken commands to Alexa.

Aim: The aim of this project is to show how Alexa can be used with Node-RED in a Raspberry Pi project to control a device remotely.

Block diagram: The block diagram of the project is shown in Figure 15.2.

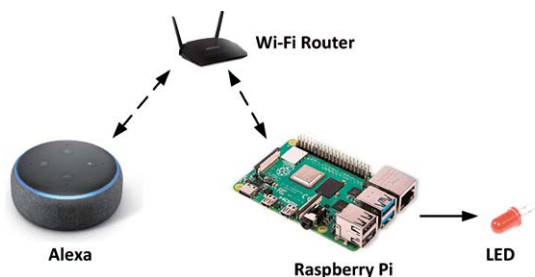


Figure 15.2 Block diagram of the project

Circuit diagram: In this project, a small LED is connected to port pin GPIO 2 of the Raspberry Pi through a 470 Ohm current limiting resistor.

Node-RED flow program: Before developing our flow program, we have to install the Alexa node to our Node Palette. Because the Alexa nodes listen on port 80, it is important that you start your Node-RED on the Raspberry Pi with a super user command. i.e,

```
pi@raspberrypi:~ $ sudo node-red-start
```

Then, the steps to install Alexa are as follows:

- Click **Menu -> Manage palette**, click **Install**
- Enter **node-red-contrib-amazon.echo** and click **install**
- You should see two new nodes called **amazon echo hub** and **amazon echo device** added to your Node Palette

If you find that your Raspberry Pi GPIO nodes are not listed in the Nodes Palette, you can re-install them using the following steps:

- Click **Menu -> Manage palette**, click **Install**
- Enter **node-red-node-pi-gpio** and click **install**
- We are now ready to develop our flow program.

Figure 15.3 shows the flow program which consists of just 4 nodes: an **amazon echo hub** node, an **amazon echo device** node, a **function node**, and a **rpi gpio out** node.

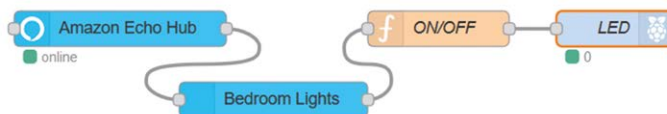


Figure 15.3 Flow program of the project

The steps are as follows:

- Create an **amazon echo hub** node and make sure that the **Port** is set to 80 and **Process Input** is set to No.
- Create an **amazon echo device** node and set the **Name** to Bedroom Lights (this is the name that Alexa will know the LED. You should choose your own name here).

- Create a **function** node and name it as ON/OFF. Enter the following statements inside this node. This node outputs 1 if the message coming from Alexa is **on**, or it outputs 0 if the message coming from Alexa is **off**:

```
if(msg.payload == "on")
  out = 1;
else
  if(msg.payload == "off")
    out = 0;
msg.payload = out;
return msg;
```

- Create an **rpi gpio out** node and name it as LED. Set the **Pin** to GPIO 2, **Type** digital output, and **Initialize** the pin state to logic 0.
- Join all the nodes as in Figure 15.3 and click **Deploy**.

To test the project, ask Alexa "**Alexa, discover devices**". After a minute or so Alexa will confirm that it has detected your device (Bedroom Lights in this project).

To turn the LED ON, ask Alexa "**Alexa, turn on bedroom lights**"

To turn the LED OFF, ask Alexa "**Alexa, turn off bedroom lights**"

The nodes can be used to turn on/off a device or to dim a light and it is supported by the 1st, 2nd and 3rd generation Alexa devices. Notice that we can add as many **amazon echo devices** as we wish to our design for the control of various equipment. An example project is given below where we have added a buzzer to our design.

15.3 Project 74 – Controlling an LED and a buzzer using Alexa

Description: In this project, an LED and a buzzer are connected to the Raspberry Pi GPIO ports. The LED and the buzzer are turned ON and OFF by giving spoken commands to Alexa.

Aim: The aim of this project is to show how multiple devices can be controlled by Alexa.

Block diagram: The block diagram of the project is shown in Figure 15.4.

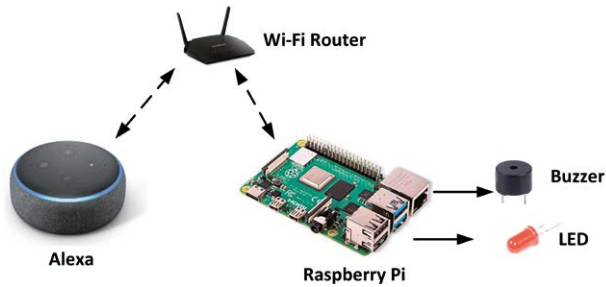


Figure 15.4 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 15.5, where the LED is connected to GPIO 2 and the buzzer is connected to GPIO 3.

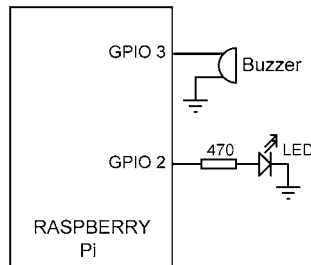


Figure 15.5 Circuit diagram of the project

Node-RED flow program: Figure 15.6 shows the flow program. Notice that we have added another **amazon echo device** node to our program and named it as **Buzzer**. This second node drives another function node which has exactly the same content as the one in Figure 15.3. The second **rpi gpio out** node is used to drive the buzzer where its **Pin** is set to GPIO 3 and its **Type** is digital output as before. Because the new amazon echo device is not known to Alexa, we have to ask Alexa to discover nodes again: "**Alexa, discover devices**". After this, we can control the LED and the buzzer by asking the Alexa:

To turn the LED ON, ask Alexa "**Alexa, turn on bedroom lights**"

To turn the LED OFF, ask Alexa "**Alexa, turn off bedroom lights**"

To turn the Buzzer ON, ask Alexa "**Alexa, turn on buzzer**"

To turn the Buzzer OFF, ask Alexa "**Alexa, turn off buzzer**"

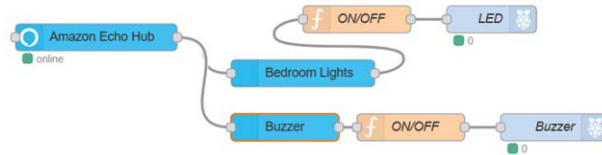


Figure 15.6 Flow program of the project

Notice that we could have used another Alexa node called **node-red-contrib-alexa-home-skill** for more sophisticated Alexa based control tasks. This node, called the Alexa Home node, requires the user to be registered and have an account before using it. Alexa Home node supports the following commands (Interested readers can get much more information from the web link: <https://alexa-node-red.bm.hardill.me.uk/docs>):

- TurnOnRequest
- TurnOffRequest
- SetPercentageRequest
- IncrementPercentageRequest
- DecrementPercentageRequest
- SetTargetTemperatureRequest
- IncrementTargetTemperatureRequest
- DecrementTargetTemperatureRequest
- GetTemperatureReadingRequest
- GetTargetTemperatureRequest
- SetLockState
- GetLockState
- SetColorRequest
- SetColorTemperatureRequest

Modified program

In the previous project, the LED is turned ON and OFF by sending spoken commands to Alexa. In some applications we may want to turn the LED ON by sensing a command, then we may want the LED to turn OFF automatically after some time. For example, before entering a room we may ask Alexa to turn the light ON and after 20 seconds we may want the light to turn OFF automatically. This can be done easily by adding a **trigger** node to Figure 15.3. In the example flow program shown in Figure 15.7, the LED is turned OFF after 15 seconds. The trigger node is configured to send 1 for 15 seconds and then it sends 0. Figure 15.8 shows the **trigger** node contents. Notice that the output of the **trigger** node is reset if msg.payload is equal to 0 (i.e. if Alexa is asked to turn the Bedroom Lights OFF).

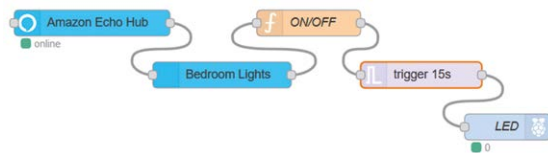


Figure 15.7 Modified flow program

The image shows the configuration interface for a 'trigger' node in Node-RED. The configuration is as follows:

- Send:** A dropdown menu showing 'a' and a value of '1'.
- then:** A dropdown menu showing 'wait for'.
- Delay:** A text input field containing '15' and a unit dropdown menu showing 'Seconds'.
- extend delay if new message arrives:** An unchecked checkbox.
- then send:** A dropdown menu showing 'a' and a value of '0'.
- Reset the trigger if:** A list of conditions:
 - msg.reset is set
 - msg.payload equals 0
- Handling:** A dropdown menu showing 'all messages'.

Figure 15.8 Configure the trigger node

15.4 Summary

In this chapter, we have learned how to use Amazon Alexa with Node-RED to control equipment connected to a Raspberry Pi. Although the projects in this chapter use the Raspberry Pi, there are no reasons why Arduino or the ESP32 DevKitC cannot be used in Alexa based projects. As described in the earlier chapters, we can connect Arduino or the ESP32 DevKitC to Raspberry Pi and then control their GPIO ports via the Raspberry Pi by sending commands via Alexa.

In the next chapter, we will be learning how to access Node-RED on Raspberry Pi from anywhere on earth.

Chapter 16 • Accessing Raspberry Pi Node-RED from anywhere

16.1 Overview

In all of the previous projects, we have been accessing Raspberry Pi Node-RED over a Wi-Fi link through our Wi-Fi router. There are many cases where we may want to access Node-RED from anywhere on earth, so that for example appliances in our house can be controlled and monitored remotely from anywhere on earth. In this chapter, we will learn how this can be done.

16.2 The ngrok

We will be using a free service called **ngrok** in order to create a tunnel to our Raspberry Pi so that we can access it from anywhere in the world. First of all, we have to change our default port number from 80 to something else, e.g. 1880 since for some reason ngrok does not work with port number 80.

The steps to install and use **ngrok** are as follows:

- Go to the website: <https://ngrok.com>
- Enter your details as shown in Figure 16.1 to create an account.

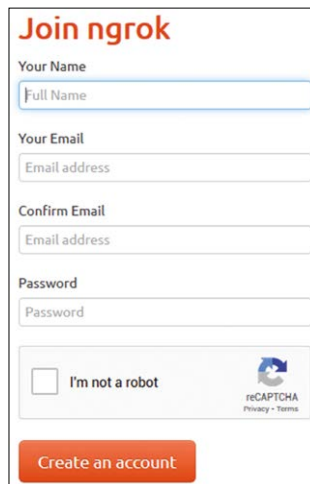
The image shows a web form titled "Join ngrok" in orange text. Below the title are four input fields: "Your Name" with a placeholder "Full Name", "Your Email" with a placeholder "Email address", "Confirm Email" with a placeholder "Email address", and "Password" with a placeholder "Password". Below these fields is a checkbox labeled "I'm not a robot" next to a reCAPTCHA logo and the text "reCAPTCHA Privacy - Terms". At the bottom of the form is an orange button labeled "Create an account".

Figure 16.1 Enter your details

- Click **Auth** link at the left-hand side to get your unique Tunnel token as shown in Figure 16.2. In this example author's tunnel token is (this is modified for security reasons): **gqudoMhDuJeXnLqVGHqv7jyUTtbYpnQzTk3Eb**

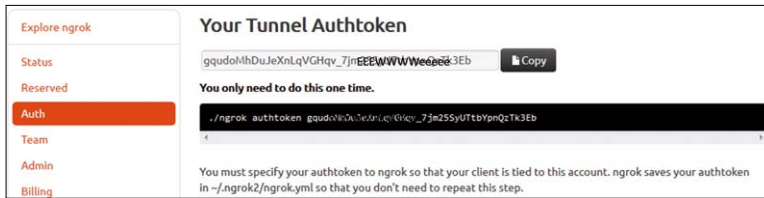


Figure 16.2 Tunnel token

- Select the **Linux (ARM)** version of the software
- Copy the software to a folder on your Raspberry Pi (e.g. **/home/pi**). You can if you wish, download the software to your PC and then use the WinSCP utility to copy it to your Raspberry Pi. At the time of writing this book, the software was named: **ngrok-stable-linux-arm.zip**
- Unzip the software on your Raspberry Pi and save the token in file **ngrok.yml** by entering the command shown in Figure 16.3

```
pi@raspberrypi:~ $ unzip ngrok-stable-linux-arm.zip
Archive: ngrok-stable-linux-arm.zip
  inflating: ngrok
pi@raspberrypi:~ $ ./ngrok authtoken gqudoMhDuJeXnLqVGHqv_7JmEEWWWWWe6003Eb
Authtoken saved to configuration file: /home/pi/.ngrok2/ngrok.yml
pi@raspberrypi:~ $
```

Figure 16.3 Unzip the software

- Create a secure tunnel with port number 1880:

```
pi@raspberrypi:~ $ ./ngrok http 1880
```

- You will be presented with a screen similar to the one shown in Figure 16.4. Here we want the part that starts with **Forwarding** and **https**, as this is the address we will be entering into our web browser, i.e.

<https://2353edf8.ngrok.io>

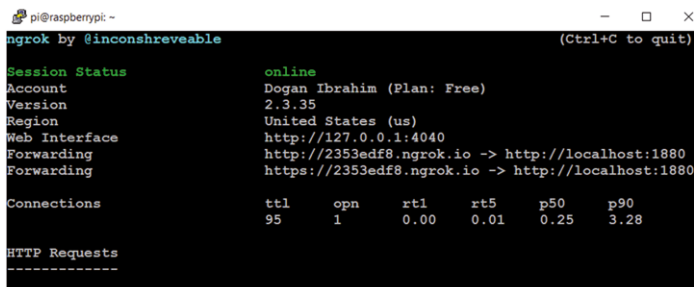


Figure 16.4 Raspberry Pi screen

- Start Node-RED on your Raspberry Pi manually (or automatically during the re-boot time. This is shown in the next section of this Chapter)

- You can now access Node-RED remotely by entering the following address into your web browser. Note that you must not close the ngrok screen (Figure 16.4) while accessing Node-RED remotely:

<https://2353edf8.ngrok.io>

16.3 Starting Node-RED automatically at reboot time

The following command can be given to automatically start Node-RED after your Raspberry Pi is started:

```
pi@raspberrypi:~ $ sudo systemctl enable nodered.service
```

Similarly, to disable the starting of Node-RED automatically at reboot time, enter:

```
pi@raspberrypi:~ $ sudo systemctl enable nodered.service
```

You can enter the following command to check if Node-RED is running:

```
pi@raspberrypi:~ $ ps axg | grep node-red
```

16.4 Summary

In this chapter, we have learned how to access Node-RED from anywhere on Earth.

In the next chapter, we will be looking at how Bluetooth communication can be used with Node-RED based projects on Raspberry Pi.

Chapter 17 • Using Bluetooth with Node-RED

17.1 Overview

Bluetooth is a wireless communications technology used over short distances. Although it was originally developed as a wireless alternative to RS232 based serial communication, it is widely used nowadays to transfer data between two Bluetooth compatible devices. Most smart mobile phones have built-in Bluetooth hardware and they can be used to transfer data, e.g. pictures to other PCs and to other compatible mobile phones. Bluetooth operates in the 2.4 to 2.485 GHz frequency range and is managed by the Bluetooth Special Interest Group.

In this chapter, we will learn how to use Bluetooth with Node-RED on a Raspberry Pi. Before using Bluetooth with Node-RED we have to install it in the Node Palette. The steps are:

- Start Node-RED
- Click **Menu -> Manage palette** and click **Install**
- Enter **node-red-contrib-bluetooth-serial-port** and click **install**
- You should see two new nodes named **bt serial in** and **bt serial out** added to your Node Palette

Now we have to install Bluetooth on our Raspberry Pi:

```
pi@raspberrypi:~ $ sudo apt-get install bluez
```

An example project is given in the next section to show how the Bluetooth node can be used.

17.2 Project 75 – Controlling an LED and a buzzer using Bluetooth

Description: In this project, an LED and a buzzer are connected to the Raspberry Pi GPIO ports. The LED and the buzzer are turned ON and OFF by sending commands from a smart mobile phone via the Bluetooth interface.

Aim: The aim of this project is to show how Bluetooth can be used with Node-RED on a Raspberry Pi.

Block diagram: The block diagram of the project is shown in Figure 17.1.



Figure 17.1 Block diagram of the project

Circuit diagram: The circuit diagram of the project is as shown in Figure 15.5, where the LED is connected to GPIO 2 and the buzzer is connected to GPIO 3 of Raspberry Pi.

Node-RED flow program: Figure 17.2 shows the flow program of the project which consists of 4 nodes: a **bt serial in** to receive data from the Bluetooth device, a **function** node to format the data for the GPIO ports, and two **rpi gpio out** nodes to control the LED and the buzzer.

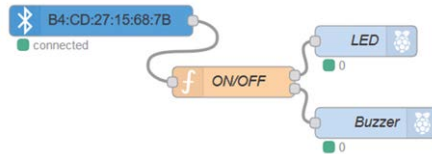


Figure 17.2 Flow program of the project

The steps are as follows:

- Create a **bt serial in** node. In this project, we are using an Android mobile phone. We have to find the Bluetooth MAC address of our mobile phone and enter it into the **bt serial in** node. The MAC address of the Android mobile phone can be found as follows:
- Enable Bluetooth on your Android mobile phone
- Click **Settings**
- Click **System**
- Click **About phone**
- Click **Status**
- Scroll down until you see the Bluetooth address of your Android mobile phone (see Figure 17.4). The MAC address of the mobile phone in this project was B4:CD:27:15:68:7B

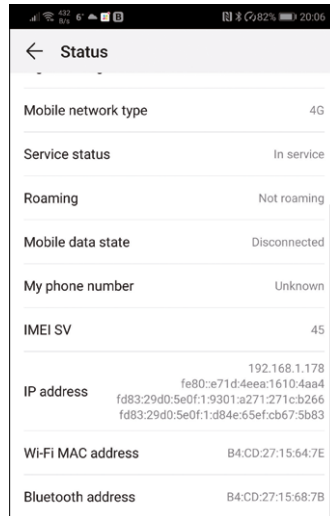


Figure 17.3 Bluetooth MAC address of the Android mobile phone

Figure 17.4 shows how the **bt serial in** node should be configured.



Figure 17.4 Configure node **bt serial in**

- Create a **function** node named **ON/OFF** and enter the following data into this node. The data returned by the **bt serial in** node is in binary and it is converted into string and stored in variable **T**. If the command is LON then output 1 is set to 1, if the command is LOFF then output 1 is set to 0. Similarly, output 2 is set to 1 and 0 if the commands are BON and BOFF respectively:

```
var out1 = null;
var out2 = null;
var T = msg.payload.toString();
if(T == "LON")
    out1 = {payload: 1};
else if(T == "LOFF")
    out1 = {payload: 0};

if(T == "BON")
    out2 = {payload: 1};
else if(T == "BOFF")
```

```
out2 = {payload: 0};
return[out1, out2];
```

- Create two **rpi gpio out** nodes named **LED** and **Buzzer**. Set the **Pin** to GPIO 2 for the LED and to GPIO 3 for the Buzzer. Set the **Type** to digital output, and **Initialize** the pin to 0 for both nodes.
- Join all the nodes as in Figure 17.3, and click **Deploy**

We can send data to our Raspberry Pi using a Bluetooth app on our Android mobile phone. There are many Bluetooth apps in the Play Store. The one used in this project is called **Bluetooth Terminal** by *Qwerty* as shown in Figure 17.5. Install this app on your mobile phone.

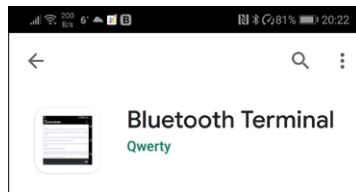


Figure 17.5 Bluetooth Terminal apps

In order to be able to access the Raspberry Pi from a mobile phone app, make the following changes to your Raspberry Pi from the command line:

- Start nano to edit the following file:

```
pi@raspberrypi:~ $ sudo nano /etc/systemd/system/dbus-org.bluez.service
```

- Add **-C** at the end of the **ExecStart=** line. Also add another line after the **ExecStart** line. The final two lines should look like:

```
ExecStart=/usr/lib/bluetooth/bluetoothd -C
ExecStartPost=/usr/bin/sdptool add SP
```

- Exit and save the file by entering **Ctrl+X** and **Y**
- Reboot the Raspberry Pi 3:

```
pi@raspberrypi:~ $ sudo reboot
```

Before sending a command to our Raspberry Pi we need to know the Bluetooth MAC address of our Raspberry Pi. This is found by entering the following in the command mode (your device name will probably be different):


```
pi@raspberrypi:~ $ bluetoothctl
[MyDevice]# show
```

You will find the Bluetooth MAC address listed at the top of the list. Enter the following commands to make the Bluetooth discoverable on your Raspberry Pi:

```
[MyDevice]# agent on
[MyDevice]# discoverable on
[MyDevice]# exit
```

Now, make sure that Node-RED is running. Start the Bluetooth apps on your mobile phone. And click the 3 dots at the top right-hand side. Click **Make discoverable**, then click **Connect a device – Insecure**, select the Raspberry Pi, and wait until the message **connect-ed: raspberrypi** is displayed. You will also see that the message **connected** is displayed under node **bt serial in** in your flow program. You are now ready to send commands from your mobile phone to the Raspberry Pi over the Bluetooth link. Send the following commands (see Figure 17.6):

LON	to turn ON the LED
LOFF	to turn OFF the LED
BON	to turn ON buzzer
BOFF	to turn OFF buzzer

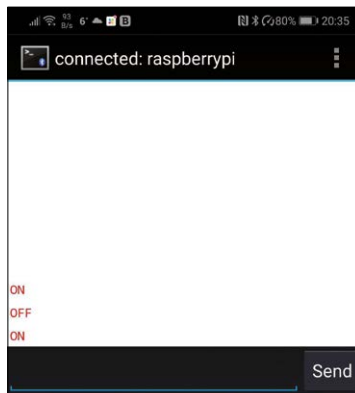


Figure 17.6 Send commands to control the LED

In this project, we have received commands from the mobile phone over a Bluetooth link. You may like to try and use the **bt serial out** node to send commands from the Raspberry Pi back to the mobile phone using the Bluetooth link.

17.3 Summary

In this chapter, we have learned how to communicate using Bluetooth Node-RED nodes. We have developed a project where an LED connected to Raspberry Pi can be controlled by sending commands from a mobile phone using the Bluetooth link. It is possible to use an Arduino development board or an ESP32 DevKitC together with a Raspberry Pi in order

to control devices connected to Arduino or the ESP32 DevKitC using Bluetooth connectivity with Node-RED.

In the next chapter, we will be looking at the important topic of MQTT.

Chapter 18 • Node-RED and MQTT

18.1 Overview

MQTT stands for **M**essage **Q**ueuing **T**elemetry **T**ransport. It is a message protocol that was created for machine-to-machine (M2M) communication, and is based on publishing and subscribing. MQTT is particularly very useful when it comes to sending data and controlling information with low bandwidth and high response times. This protocol is especially worth considering when it is required to send data to actuators and retrieve data from sensors. MQTT was first developed by IBM in 1999 mainly for satellite communication, and since then has become the standard communications protocol for the Internet of Things (IoT) applications. MQTT uses your existing home network to send messages to your IoT devices and receive a response from them. The official website of MQTT is: <http://mqtt.org>

MQTT works on top of the TCP/IP protocol and it is faster than sending HTTP requests since a message can be as small as 2 bytes and there are no headers which is the case with the HTTP. Additionally, in MQTT messages are distributed automatically to the interested clients.

18.2 How MQTT works

It is worthwhile to learn how MQTT works before it is used in projects. In MQTT protocol, there is a sender called the **Publisher**, and a receiver called the **Subscriber**. Between these two, a server called a **Broker** is used to act as the intermediary. Figure 18.1 shows the basic structure of an MQTT based system.

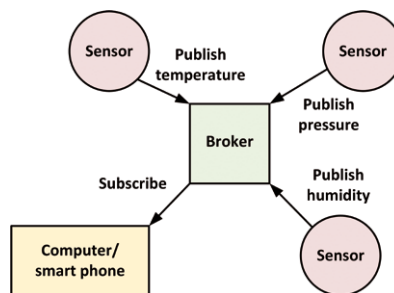


Figure 18.1 Basic MQTT structure

In Figure 18.1, the temperature sensor publishes (or sends) its measured value to the broker, who accepts and saves the data. The broker then sends this measured value to other devices that have subscribed to receive this data.

In MQTT there are 5 basic concepts that you should understand:

- Publisher
- Subscriber
- Messages
- Topics
- Broker

Publisher: The publisher is the node that sends data or the message on a topic. For example, a publisher can be a sensor node that sends ambient temperature values.

Subscriber: The subscriber can be a computer, a smartphone, a microcontroller or any other processor that subscribes to receive the data or the message. For example, a device publishes on a topic, another device subscribes to the same topic and receives the published message.

Messages: Messages are the information exchanged in the MQTT network. A message can be data or a command. For example, a message can be the temperature value, or a command to turn ON a switch.

Topics: Topics are important concepts of MQTT. These are the ways we register interest for incoming messages, or how we specify where we want to publish the message. i.e. the message exchange takes place via the topics. Topics in MQTT are represented with character strings, separated by a forward slash. Each forward slash indicates a topic level. An example is shown below which creates a topic for an LED in your kitchen:

home/kitchen/led

In the above example, home, kitchen, and led are the topic levels and they are separated by the topic level separators. The topics are case sensitive and for example, Home/kitchen/led is not the same as above.

Single level wildcards: We can use wildcards in topics so that several sensors can be queried at the same time. A "+" sign is used to identify a single level wildcard. Example is given below:

home/+/led	topic with single level wildcard
home/bedroom/led	wildcard replaced with bedroom
home/livingroom/led	wildcard replaced with livingroom
home/garden/led	wildcard replaced with garden

Multi-level wildcards: We can also use a multi-level wildcard, represented by the sign "#". This must always be used at the end of a topic. An example is given below:

home/kitchen/#	topic with multi-level wildcard
home/bedroom/led	kitchen replaced with bedroom
home/livingroom/light	livingroom and light replaced

Broker: the broker receives all messages, filters them and decides who are interested in them, and then sends the message to all subscribed clients

Figure 18.2 illustrates the operation of MQTT, which can be summarised as follows:

- The processor (and sensor) publishes ON and OFF messages on-topic **home/kitchen/led**
- We have a Raspberry Pi that controls an LED. The Raspberry Pi is subscribed to topic: **/home/kitchen/led**
- When a new message is published on topic **home/kitchen/led**, the Raspberry Pi receives the ON or OFF messages and turns the LED ON or OFF.

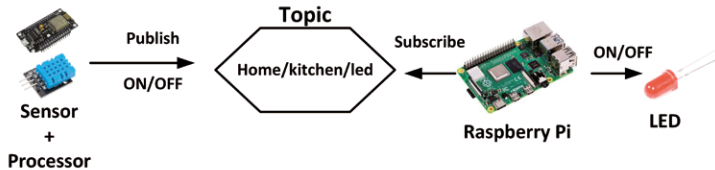


Figure 18.2 Operation of MQTT

18.3 The Mosquitto broker

The broker is one of the important parts of MQTT. There are several brokers that one can use. In most home automation projects the **Mosquitto** broker is used. The steps to install this broker on the Raspberry Pi is as follows:

```

pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get install mosquitto mosquitto-clients
  
```

We can make Mosquitto to auto-start on boot by entering the following command:

```

pi@raspberrypi:~ $ sudo systemctl enable mosquitto.service
  
```

To test the Mosquitto installation, enter the command **mosquitto -v** as shown in Figure 18.3. This command displays the version of the Mosquitto running on your system (Don't worry about the Error: Address already in use).

```

pi@raspberrypi:~ $ mosquitto -v
1577732396: mosquitto version 1.5.7 starting
1577732396: Using default config.
1577732396: Opening ipv4 listen socket on port 1883.
1577732396: Error: Address already in use
pi@raspberrypi:~ $ █
  
```

Figure 18.3 Testing the Mosquitto installation

We can test if our broker is working correctly as follows:

Subscribe to the topic "TestTopic" by entering the following command. Mosquitto_sub tells that we want to subscribe to a topic, and the name following **-t** is the topic name. Now, every time we publish (i.e. send message) to TestTopic, the message will appear in the window:

```
pi@raspberrypi:~ $ mosquitto_sub -t "TestTopic"
```

Now, because the terminal is listening for messages from our broker, we have to open another window so that we can publish messages. After opening another window, enter the following command to publish to TestTopic:

```
pi@raspberrypi:~ $ mosquitto_pub -t "TestTopic" -m "Hello There!"
```

Here, the topic name is after `-t`, and the message is after `-m`. After hitting the Enter we will see the message appear on our subscriber terminal as shown in Figure 18.4.

```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ mosquitto_sub -t "TestTopic"
Hello There!

pi@raspberrypi:~ $ mosquitto_pub -t "TestTopic" -m "Hello There!"
pi@raspberrypi:~ $
```

Figure 18.4 Broker example

18.4 Using MQTT in home automation and in IoT projects

In order to use MQTT in home automation and in IoT projects we need the following:

- A Raspberry Pi
- Node-RED and MQTT
- An Arduino, ESP32, ESP8266, or any other compatible microcontroller

Figure 18.5 shows the basic MQTT system setup. The sensors, actuators, relays, switches, lamps, LEDs, etc are connected to the system via an Arduino, ESP32, ESP8266, or any other compatible processor.

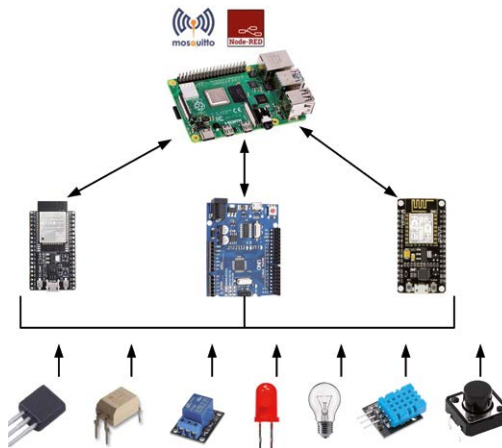


Figure 18.5 MQTT system setup

Node-RED offers two MQTT nodes (**mqtt in** and **mqtt out**) which can be found in the network palette as shown in Figure 18.6.

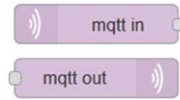


Figure 18.6 MQTT nodes

We can send a message to the MQTT broker using an **inject** node as shown in Figure 18.7. Notice that the small green box under the **mqtt out** node indicates that the connection from the **mqtt out** node to the broker has been established. In Figure 18.7 the topic is set to **TestTopic**, and the **inject** node **Payload** is set to message **Hello From Me!!**. This message is displayed on the MQTT subscriber as shown in the figure. In this flow program, the **mqtt out** node **Server** was set to **localhost** and the **Port** number was set to 1883 (network ports 1883 and 8883 are reserved by default):

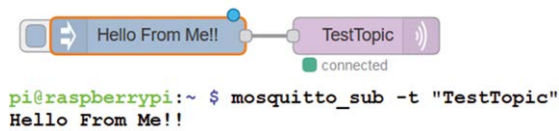


Figure 18.7 Sending a message to the MQTT broker

Notice that, by using different topics we can send (publish) different messages and these messages will be received by the subscribers having the same topics.

A very simple example is given below which illustrates how an LED can be controlled using MQTT.

18.5 Project 76 – Controlling an LED using MQTT

Description: In this project, we will learn how an LED can be controlled using MQTT.

Aim: The aim of this project is to show how MQTT can be used in a very simple project.

Circuit diagram: In this project, an LED is connected to Raspberry Pi port pin GPIO 2 through a current limiting resistor.

Node-RED flow program: Figure 18.8 shows the flow program of the project. Basically, two **inject** nodes are used with the Payloads set to 1 and 0 respectively. When for example the button of the upper **inject** node is clicked then a 1 is sent to the broker via the **mqtt out** node. The **mqtt in** node in the lower flow is given the same topic name as the **mqtt out** node, and this node drives the **rpi gpio out** node which controls the LED accordingly. The Raspberry Pi console displays the data received by the GPIO port since it is subscribed to the same topic as shown in Figure 18.9.

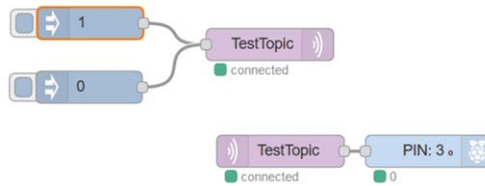


Figure 18.8 Flow program of the project

```
pi@raspberrypi:~ $ mosquitto_sub -t "TestTopic"
1
0
1
0
```

Figure 18.9 Raspberry Pi console subscribed to the same topic

In more complex and real projects instead of using inject nodes, we usually have processors with sensors connected to their inputs. The data from these sensors are sent to the broker via the **mqtt out** node. The mqtt nodes then receive and process this data (e.g. display or activate an actuator).

In the next sections, we will learn how to use the ESP8266 NodeMCU development board as the client in an MQTT application.

18.6 The ESP8266 processor

The ESP8266 is a low-cost Wi-Fi chip that can be used with or without a microcontroller to access a Wi-Fi network. The chip has been developed by Shanghai-based Chinese manufacturer Espressif Systems and incorporates full TCP/IP stack. It can be used as a small microcontroller on its own, or it can be interfaced with a bigger microcontroller development system such as the Arduino where it can be used to provide Wi-Fi capability to the host system.

There are many versions of the ESP8266 chip, available as modules on small boards with different functionalities and different input/output capacities. The basic specifications of the ESP8266 chip are summarized below:

- 32-bit RISC CPU: Tensilica Xtensa LX106
- Operation speed 80 to 160 MHz (by overclocking)
- 3.3V operation (+2.5V min to +3.6V max)
- 64 Kbytes program memory, 96 Kbytes data memory
- External QSPI flash memory (up to 16 Mbytes)
- IEEE 802.11 b/g/n Wi-Fi support
- WEP or WPA/WPA2 authentication (or, open network)
- 1 x 10-bit analog-to-digital (ADC) converter
- Up to 16 GPIOs (not +5V tolerant)
- Shared SDIO, SPI, I²C, I2S interface pins
- Integrated TCP/IP protocol stack
- UART on dedicated pins (additional TX only UART on GPIO2 pin)
- Standby power < 10 mW

NodeMCU (LoLin) is a small microcontroller development board based on the ESP8266 processor. It was first manufactured in 2014 and currently, it is one of the most popular Wi-Fi development kits based on the ESP8266 chip and it incorporates an on-board USB-to-TTL adapter, micro USB socket, a large number of GPIOs, PWM, I²C, ADC, +3.3V voltage regulator, UART and so on. Figure 18.10 shows the pin configuration of the NodeMCU.

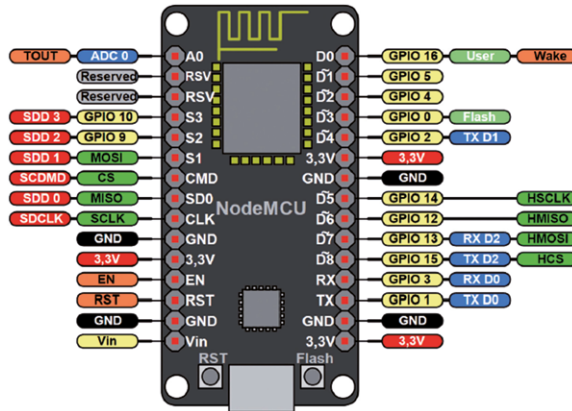


Figure 18.10 Pin configuration of the NodeMCU

NodeMCU can be programmed using the Arduino IDE. The steps to install the necessary software on the Arduino IDE are given below:

- Start your Arduino IDE
- Click **File -> Preferences** and enter the following at the bottom of the screen as shown in Figure 18.11. If you already have another URL there, separate them with a comma, and click **OK**

http://arduino.esp8266.com/stable/package_esp8266com_index.json

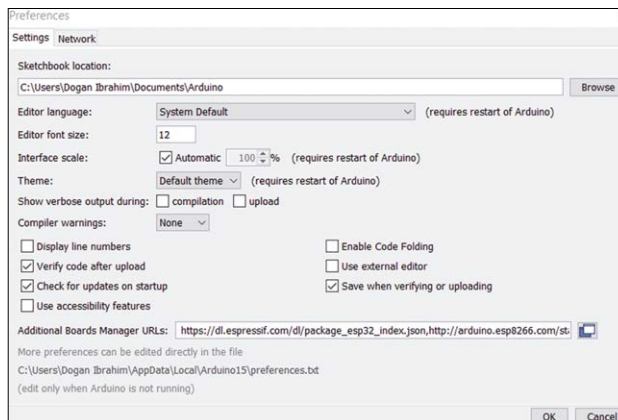


Figure 18.11 Installing the ESP8266 on Arduino IDE

- Select **Tools -> Board "Arduino/Genuino Uno" -> Boards Manager**
- Enter **esp8266** and press the **install** button for **ESP8266 by ESP8266 Community** (see Figure 18.12)



Figure 18.12 Enter esp8266

- After a successful installation, you should see the Installed message as in Figure 18.13. Close the screen.

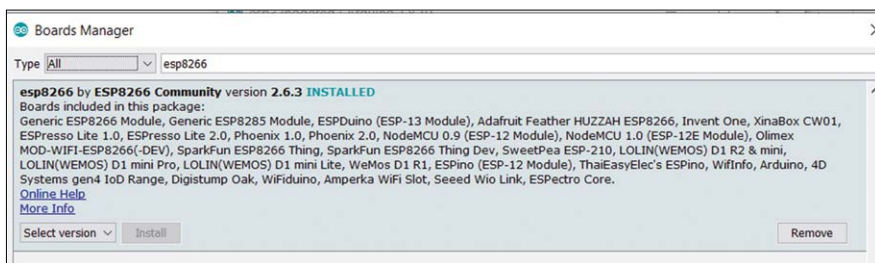


Figure 18.13 Successful installation

Detailed programming of the ESP8266 processor is beyond the scope of this book. A simple project is given in the next section just to remind the users of how the ESP8266 can be programmed using the Arduino IDE. It is assumed that the readers are familiar with using the Arduino IDE.

18.7 Project 77 – Flashing an LED using ESP8266 NodeMCU

Description: This is a very simple project where an LED is connected to one of the GPIO ports of the NodeMCU and the LED is flashed every second.

Aim: The aim of this project is to remind the users of how the NodeMCU can be programmed using the Arduino IDE.

Circuit diagram: Figure 18.14 shows the circuit diagram of the project. The LED is connected to GPIO port 16 of the NodeMCU.

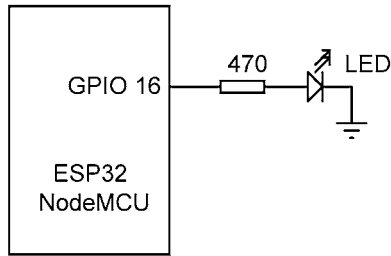


Figure 18.14 Circuit diagram of the project

Program Listing: The program listing (program: **esp8266led**) is shown in Figure 18.15. The steps to start programming are as follows:

Build the circuit and connect the NodeMCU to your PC through a mini USB cable

- Start Arduino IDE
- Click **Tools -> Board -> NodeMCU 1.0 (ESP-12E Module)**
- Select the COM port and click **Tools -> Port n** where n is the port number
- Enter the program and save it with the name **esp8266led**:

```
int LED = 16;
void setup()
{
    pinMode(LED, OUTPUT);
}

void loop()
{
    digitalWrite(LED, HIGH);    // LED ON
    delay(1000);                // Wait one second
    digitalWrite(LED, LOW);     // LED OFF
    delay(1000);                // Wait one second
}
```

Figure 18.15 program esp8266led

- Click **Sketch -> Verify/Compiler** and make sure there are no compilation errors
- Click **Sketch -> Upload** and wait until you see the message **Done uploading**.
- You should see the LED flashing every second

Figure 18.16 shows the circuit built on a breadboard.

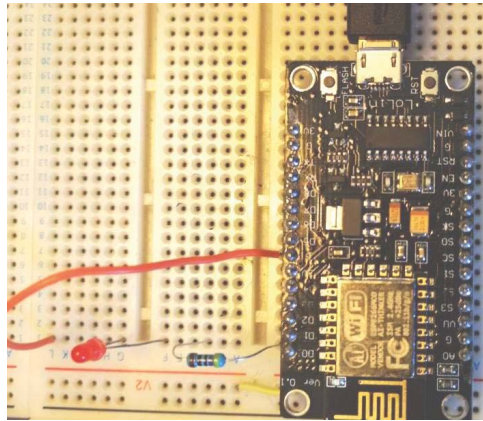


Figure 18.16 The circuit built on a breadboard

18.8 Using the ESP8266 NodeMCU with MQTT

In order to get MQTT working on ESP8266, we need to install a library called **PubSubClient**. This library can be downloaded from the following link;

<https://github.com/knolleary/pubsubclient/archive/master.zip>

The steps are as follows:

- Download the above **.zip** file (Click **Clone or Download** and select **Download ZIP**). The file is named: **pubsubclient-master.zip**
- Start Arduino IDE
- Click **Sketch -> Include Library -> Add .ZIP Library**
- Select the path to the **.zip** file
- Select **File -> Examples -> PubSubClient** and you should see a number of mqtt based example files listed

You should make Mosquitto auto-start on boot by entering the following command:

```
pi@raspberrypi:~ $ sudo systemctl enable mosquitto.service
```

A simple project is given below to illustrate how MQTT can be used with the ESP8266 NodeMCU.

18.9 Project 78 – Controlling an LED using ESP8266 NodeMCU with MQTT – LED connected to Raspberry Pi

Description: This is a very simple project where an LED is connected to one of the GPIO ports of the Raspberry Pi. Additionally, a push-button switch is connected to the ESP8266 NodeMCU. The LED is toggled when the button is pressed.

Aim: The aim of this project is to show how MQTT can be used with the ESP8266 NodeMCU. Here, the NodeMCU is the client (publisher) and the Raspberry Pi is the subscriber which receives the messages.

Block diagram: Figure 18.17 shows the block diagram of the project.

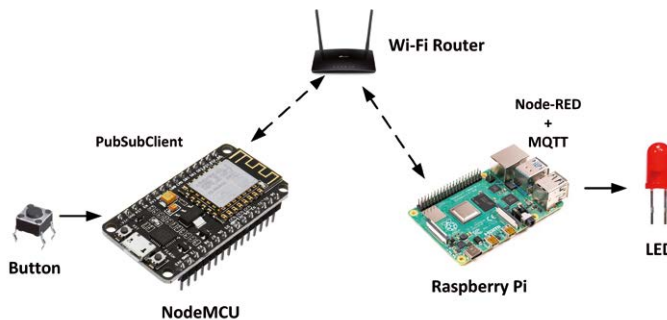


Figure 18.17 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 18.18. The button is connected to port pin GPIO 2 of the ESP8266 NodeMCU. Similarly, the LED is connected to port pin GPIO 2 of the Raspberry Pi.

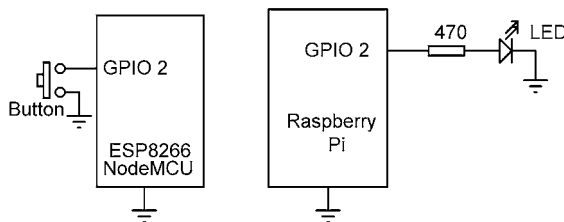


Figure 18.18 Circuit diagram of the project

ESP8266 NodeMCU program listing: Figure 18.19 shows the program listing (program: `mqttesp8266`). NodeMCU uses the Wi-Fi link to communicate with the Raspberry Pi in MQTT based applications. Therefore, we need to know the SSID name and the password of our Wi-Fi router, and the IP address of our Raspberry Pi. In this example we have the following settings (you will have to enter your own settings):

SSID: BTHomeSpot-XNH
 Password: 49350baeb
 Raspberry Pi IP address: 192.168.1.202

We have to enter these values into our program. Notice that the button is connected to port pin GPIO 2:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "BTHomeSpot-XNH";
const char* password = "49350baeb";
const char* mqtt_server = "192.168.1.202";
const byte Button = 2;
const char *ID = "Example_Button";
bool SwitchState = 0;
```

In this example, we have chosen the MQTT topic as **kitchen/light**

```
const char *Topic = "kitchen/light/"
```

We have to set up a Wi-Fi client, and a PubSubClient:

```
WiFiClient wclient;
PubSubClient client(wclient);
```

Inside the setup routine, we will initialize the Arduino IDE Serial Monitor at 115200 Baud so that we can see trace the execution of the code easily. We will also have to configure port pin GPIO 2 where the switch is connected to as input and enable the pull-up resistor, and enable the Wi-Fi on our ESP8266 NodeMCU:

```
//
// The setup routine
//
void setup()
{
  Serial.begin(115200);
  pinMode(Button, INPUT);
  digitalWrite(Button, HIGH);
  delay(100);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}
```

The Wi-Fi is setup inside the setup routine as follows (readers who used the ESP8266 before should be familiar with the following code):

```
//
// Connect to local Wi-Fi router
//
void setup_wifi()
{

    Serial.print("\nESP32 NodeMCU connecting to: ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to Wi-Fi");
}
```

The main program loop attempts to reconnect to the client by calling function **reconnect** if the connection is lost, and reads the status of the button to check if it is pressed.

The reconnect code is:

```
//
// Reconnect to client if the connection is lost
//
void reconnect()
{
    while (!client.connected())
    {
        Serial.print("Attempting MQTT connection...");
        if (client.connect(ID))
        {
            Serial.println("Connected");
            Serial.print("Publishing to: ");
            Serial.println(Topic);
            Serial.println('\n');
        }
        else
        {
            Serial.println(" Trying to connect again in 5 seconds");
            delay(5000);
        }
    }
}
```

The main loop code is:

```
void loop()
{
    if (!client.connected()) reconnect();
    client.loop();

    if(digitalRead(Button) == 0)    // If button is pressed
    {
        ButtonState = !ButtonState;    // Toggle ButtonState
        if(ButtonState == 1)    // ON
        {
            client.publish(Topic, "on");
            Serial.println((String)Topic + " is ON");
        }
        else
        {
            client.publish(Topic, "off");
            Serial.println((String)Topic + " is OFF");
        }
    }

    while(digitalRead(Button) == 0)    // Wait switch to release
    {
        yield();
        delay(20);
    }
}
```

Figure 18.19 *mqttesp8266 program listing*

Node-RED flow program: Figure 18.20 shows the flow program for the Raspberry Pi which consists of just 3 nodes: an **mqtt in** node, a **function** node, and a **rpi gpio out** node.

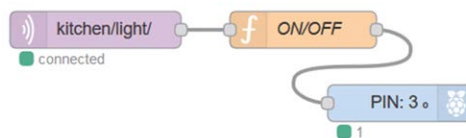


Figure 18.20 *Flow program of the project*

The steps are as follows:

- Create an **mqtt in** node and set its **Server** to localhost, **Port** to 1883, and the **Topic** to kitchen/light/

- Create a **function** node names ON/OFF and enter the following statements inside this function:

```
if(msg.payload == "on")
    msg.payload = 1;
else if(msg.payload == "off")
    msg.payload = 0;
return msg;
```

- Create a **rpi gpio out** node and set the **Pin** to GPIO 2, **Type** to digital output, and **Initialize** to 0.
- Connect all the nodes as in Figure 18.20 and click **Deploy**.

Testing

The project can be tested as follows:

- Power on the ESP8266 NodeMCU and open the Serial Monitor and set its Baud rate to 115200.
- You should see messages on the Serial Monitor saying that the NodeMCU is connected to your Wi-Fi (see Figure 18.21) and that it can publish to MQTT with the topic kitchen/light/

```
ESP32 NodeMCU connecting to: BTHomeSpot-XNH
.....
Connected to Wi-Fi
Attempting MQTT connection...connected
Publishing to: kitchen/light/
```

Figure 18.21 Connection messages on the Serial Monitor

- Press the button. You should see the following message displayed on the Serial Monitor (see Figure 18.22). At the same time the LED will turn ON:

```
kitchen/light/ is ON
```

- Release the button and press again. You should see the following message displayed on the Serial Monitor. At the same time the LED will turn OFF:

```
kitchen/light/ is OFF
```

```

ESP32 NodeMCU connecting to: BTHomeSpot-XNH
.....
Connected to Wi-Fi
Attempting MQTT connection...connected
Publishing to: kitchen/light/

kitchen/light/ is ON
kitchen/light/ is OFF
kitchen/light/ is ON
kitchen/light/ is OFF
kitchen/light/ is ON
kitchen/light/ is OFF

```

Figure 18.22 Messages on the Serial Monitor

- If you subscribe your Raspberry Pi console to topic kitchen/light/ then you should see the messages displayed there as well (see Figure 18.23)

```

pi@raspberrypi:~ $ mosquitto_sub -t "kitchen/light/"
off
on
off
on
off

```

Figure 18.23 Messages on the Raspberry Pi console

18.10 Project 79 – Controlling an LED using ESP8266 NodeMCU with MQTT – LED connected to ESP8266 NodeMCU

Description: This project is similar to the previous one but here the LED is connected to the NodeMCU and the button is connected to the Raspberry Pi. As before, the LED is toggled when the button is pressed.

Aim: The aim of this project is to show how the client (NodeMCU in this project) can be configured as a subscriber so that it can receive messages using MQTT. Raspberry Pi is a publisher in this project.

Block diagram: Figure 18.24 shows the block diagram of the project.

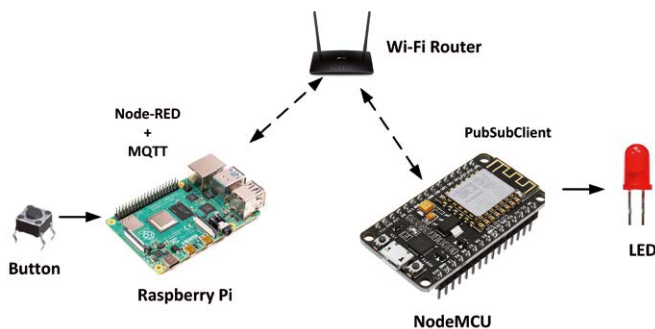


Figure 18.24 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 18.25. The button is connected to port pin GPIO 2 of the Raspberry Pi. Similarly, the LED is connected to port pin GPIO 2 of the NodeMCU through a current limiting resistor.

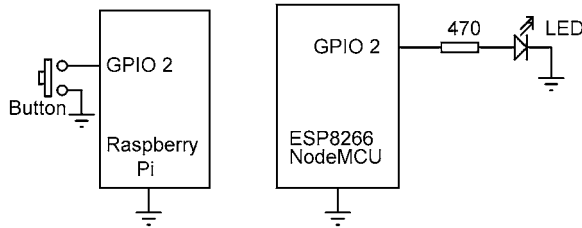


Figure 18.25 Circuit diagram of the project

ESP8266 NodeMCU program listing: Figure 18.26 shows the program listing (program: **esp8266mqtt**). The program is very similar to the one given in Figure 18.19 except the following changes:

The statement `client.subscribe(Topic)` is added to the setup routine so that the NodeMCU is configured as an MQTT subscriber.

The main program loop has been modified so that the loop only re-connects if the connection is lost.

A new function called `callback` is added to the program. This function is called every time a message is received by the NodeMCU. At the beginning of the function, the message New message arrived is displayed together with the MQTT topic name. The function then displays the received message. Character array **payload[0]** stores the first character of the received message which is either '1' or '0' in this project. If '1' is received then the LED is turned ON, otherwise, the LED is turned OFF. The contents of this function are as follows.:

```
void callback(char *topic, byte* payload, unsigned int length)
{
    Serial.print("New message arrived - Topic:");
    Serial.println(Topic);
    for(int j = 0; j < length; j++)
    {
        Serial.print((char)payload[j]);
    }
    Serial.println();
    if((char)payload[0] == '1')
    {
        digitalWrite(LED, HIGH);
    }
    else
    {
        digitalWrite(LED, LOW);
    }
}
```

```

    }
}

/*****
 *      ESP8266 NodeMCU MQTT PROGRAM
 *      =====
 * In this program a push-button switch is conencted to port
 * GPIO 2 of the Raspberry Pi. Similarly, an LED is connected
 * to port GPIO 2 of the NodeMCU. Pressing the button toggles
 * the LED.
 *
 * Author: Dogan Ibrahim
 * File  : esp8266mqtt
 * Date  : December 2020
 *
 *****/
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "BTHomeSpot-XNH";           // WiFi SSID
const char* password = "49345abaeb";           // WiFi password
const char* mqtt_server = "192.168.1.202";      // Raspberry Pi IP
const byte LED = 2;                             // LED pin
const char *ID = "Example_LED";
const char *Topic = "kitchen/light/";          // MQTT TOPIC

WiFiClient wclient;
PubSubClient client(wclient);

//
// Connect to local Wi-Fi router. This function connects the
// NodeMCU to the local WiFi router. Because the Serial Monitor
// is enabled, we can see the progress in the monitor
//
void setup_wifi()
{
    Serial.print("\nESP32 NodeMCU connecting to: ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to Wi-Fi");
}

```

```

}

//
// This is the setup routine. Here the Serial Monitor is enabled,
// Button is configured as digital input, and the WiFi is setup.
// Notice that the MQTT server (Raspberry Pi) operate on port 1883
// The callback routine is called callback
//
void setup()
{
    Serial.begin(115200);           // Enable Serial Monitor
    pinMode(LED, OUTPUT);          // Button is input
    delay(100);                    // Small delay
    setup_wifi();                  // Setup WiFi
    client.setServer(mqtt_server, 1883); // Define client
    client.setCallback(callback);
}

//
// Reconnect to client if the connection is lost
//
void reconnect()
{
    while (!client.connected())
    {
        Serial.print("Attempting MQTT connection...");
        if (client.connect(ID))
        {
            Serial.println("connected");
            Serial.print("Publishing to: ");
            Serial.println(Topic);
            Serial.println('\n');
            client.subscribe(Topic);
        }
        else
        {
            Serial.println(" Trying to connect again in 5 seconds");
            delay(5000);
        }
    }
}

//
// This is the callback routine. The program jumps here when a message

```

```

// is received by the NodeMCU (subscriber). Turn ON/OFF the LED based
// on the received message (i.e. command)
//
void callback(char *topic, byte* payload, unsigned int length)
{
    Serial.print("New message arrived - Topic:");
    Serial.println(topic);
    for(int j = 0; j < length; j++)
    {
        Serial.print((char)payload[j]);
    }
    Serial.println();
    if((char)payload[0] == '1')
    {
        digitalWrite(LED, HIGH);
    }
    else
    {
        digitalWrite(LED, LOW);
    }
}

//
// Main program loop. If client gets disconnected, re-connect
//
void loop()
{
    if (!client.connected()) reconnect();
    client.loop();
}

```

Figure 18.26 esp8266mqtt program listing

Node-RED flow program: Figure 18.27 shows the flow program for the Raspberry Pi which consists of just 3 nodes: a **rpi gpio in** node to read the button status, a **function** node, and an **mqtt out**



Figure 18.27 Flow program of the project

The steps are as follows:

- Create a **rpi gpio in** node to read the button status and set the **Pin** to GPIO 2 and **Resistor** to pullup
- Create a **function** node named **ON/OFF** and enter the following statements inside this function. The function returns "1" if the button is pressed (`msg.payload = "0"`), and "0" if the button is not pressed:

```
if(msg.payload == "0")
  msg.payload = "1";
else
  msg.payload = "0";
return msg;
```

- Create an **mqtt out** node and set its **Server** to localhost, **Port** to 1883, and the **Topic** to kitchen/light/
- Join all 3 nodes and click **Deploy**

Testing

The project can be tested as follows:

- Power on the ESP8266 NodeMCU and open the Serial Monitor and set its Baud rate to 115200.
- You should see messages on the Serial Monitor saying that the NodeMCU is connected to your Wi-Fi and that it can publish to MQTT with the topic kitchen/light/
- Press the button. You should see the following message displayed on the Serial Monitor (see Figure 18.28). At the same time the LED will turn ON:

```
New message arrived - Topic:kitchen/light/
1
```

- Release the button. Now, you should see the following message displayed on the Serial Monitor. At the same time the LED will turn OFF:

```
New message arrived - Topic:kitchen/light/
0
```

```

ESP32 NodeMCU connecting to: BTHomeSpot-XNH
.....
Connected to Wi-Fi
Attempting MQTT connection...connected
Publishing to: kitchen/light/

New message arrived - Topic:kitchen/light/
1
New message arrived - Topic:kitchen/light/
0
New message arrived - Topic:kitchen/light/
1

```

Figure 18.28 Messages on the Serial Monitor

If you publish from Raspberry Pi console to topic kitchen/light/ then you should see the LED turning ON or OFF and the messages displayed on the Serial Monitor (see Figure 18.29)

```

pi@raspberrypi:~ $ mosquitto_pub -t "kitchen/light/" -m "1"
pi@raspberrypi:~ $ mosquitto_pub -t "kitchen/light/" -m "0"
pi@raspberrypi:~ $ █

```

Figure 18.29 Publishing on the Raspberry Pi console

18.11 Summary

In this chapter, we have learned how to install and use MQTT with Node-RED. The projects given in this Chapter are based on the ESP8266 NodeMCU development board. There are no reasons why other development boards such as the ESP32 DevKitC cannot be used in MQTT based applications. This is left as an exercise for the readers.

In the next chapter, we will be learning how to use Node-RED HTTP nodes in projects.

Chapter 19 • Using HTTP in Node-RED projects

19.1 Overview

The node-red palette includes nodes for HTTP request, HTTP response, and HTTP in. The http request node is a core node that can be used for making http requests. Using this node we can retrieve web pages from a website, send and receive JSON data to a website, make API requests and so on. The following configuration is supported by the node:

- GET, POST, PUT, and DELETE methods
- URL of the resource
- Authentication
- SSL
- Response

HTTP is a text-based request and response protocol which uses the client-server model of communication. A request consists of a command, some optional headers, and optional body content. Similarly, a response consists of a status code, optional headers, and optional body content. A web browser may be the client that submits an HTTP request to the server.

Whenever we enter a URL in the address box of a browser, this URL is translated into a request message. For example, the URL <http://www.test.com/doc/index.html> may be translated into a request of the following format:

```
GET /doc/index.html HTTP/1.1
Host: www.test.com
Accept: image/gif, image/jpeg. */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
```

Here, **GET** is the method and **/doc/index.html** is the path and **HTTP/1.1** is the protocol version.

The response status code can have several values. Code 200 is the code for success.

In this section, we will learn how to use the http request node in Node-RED based projects.

19.2 Using HTTP GET

The GET request is used to request data from a specified web address. It is one of the most common HTTP methods. Node-RED node http request can be used to make a web request. An example flow program is shown in Figure 19.1 which uses node http request to make a request to the following URL (this is an example domain):

<http://example.com/hi/there?hand=wave>

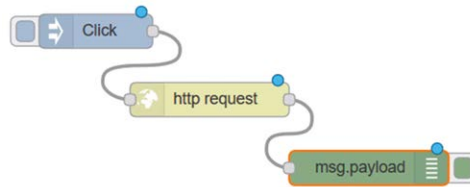


Figure 19.1 Flow program of using http request

this flow program, we have an **inject** node, an **http request** node with the **Method** is set to GET and the **URL** is set as above. A debug node is used to display the output in the Debug window. Figure 19.2 shows the output displayed.

```

03/01/2020, 23:20:00 node: 40e29d34.413454
msg.payload : string[1256]
▼ string[1256]
<!doctype html>
<html>
<head>
  <title>Example Domain</title>

  <meta charset="utf-8" />
  <meta http-equiv="Content-type"
    e="content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style type="text/css">
    body {
      background-color: #f0f0f2;
      margin: 0;
      padding: 0;
      font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
    }
  </style>
</head>
<body>
  <h1>Example Domain</h1>
  <p>This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.</p>
</body>
</html>
  
```

Figure 19.2 Output in the Debug window (only part of the output is shown)

We can insert a **html** node inbetween the **http request** node and the **debug** node and set its **Selector** to `div` to extract the `div` types, and its **Output** to display only the text content of the elements, as a single message containing an array. The output is then shown in Figure 19.3 together with the flow program.



Figure 19.3 Flow program and the output

19.3 Web server

A web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve web pages to users in response to their requests. These requests are forwarded by HTTP clients.

By using the HTTP server/client pair we can control any device connected to a web server processor over the web.

Figure 19.4 shows the structure of a web server/client setup. In this figure, Raspberry Pi is the webserver processor and the PC (or a laptop, tablet or smart mobile phone) is the web client. The device to be controlled is connected to the webserver processor. The operation of the system is as follows:

- The web server is in the listen mode, listening for requests from the web client
- The web client makes a request to the web server by sending an HTTP request
- In response, the webserver sends an HTTP code to the web client which is activated by the web browser by the user on the web client and is shown as a form on the web client screen.
- The user sends commands (e.g. ticks buttons on a web client form) and this sends code to the webserver so that the webserver can carry out the required operations.

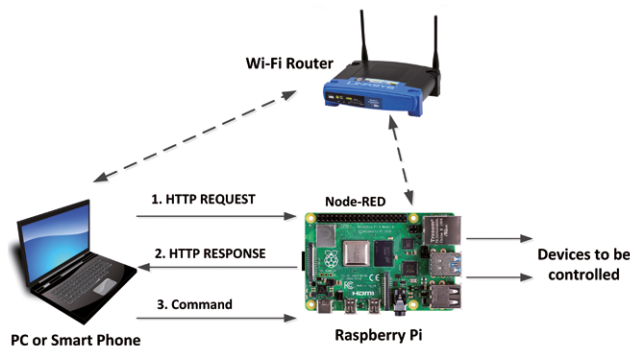


Figure 19.4 Webserver/client structure

A project is given in the next section to illustrate how a Node-RED based web server can be used to control relays.

19.4 Project 80 – Controlling 4 Relays using web server

Description: In this project, a relay module consisting of 4 relays is connected to the Raspberry Pi. These relays are activated and de-activated from a PC (or a smart mobile phone) by clicking buttons on a web browser form.

Aim: The aim of this project is to show how a web server application can be designed using Node-RED on a Raspberry Pi.

Background information: In this project, a 4-channel relay board (see Figure 19.5) from Elegoo (www.elegoo.com) is used. This is an opto-coupled relay board having 4 inputs, one for each channel. The relay inputs are at the bottom right-hand side of the board while the relay outputs are located at the top side of the board. The middle position of each relay is the common point, the connection to its left is the normally closed (NC) contact, while the connection to the right is the normally open (NO) contact. The relay contacts support AC250V at 10A and DC30V 10A. IN1, IN2, IN3, and IN4 are the **active LOW** inputs, which means that a relay is activated when a logic LOW signal is applied to its input pin. Relay contacts are normally closed (NC). Activating the relay changes the active contacts such that the common pin and NC pin become the two relay contacts and at the same time the LED at the input circuit of the relay board corresponding to the activated relay is turned ON. The VCC can be connected to either +3.3V or to +5V. Jumper JD is used to select the voltage for the relay. **Because the current drawn by a relay can be in excess of 80mA, you must remove this jumper and connect an external power supply (e.g. +5V) to pin JD-VCC.**

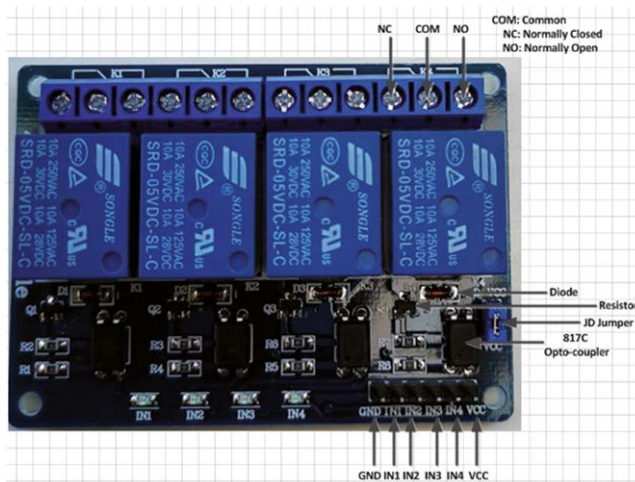


Figure 19.5 4-channel relay board

Block diagram: Figure 19.6 shows the block diagram of the project.

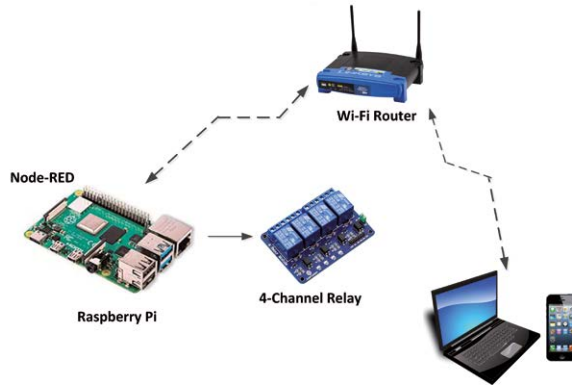


Figure 19.6 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 19.7. IN1, IN2, IN3 and IN4 inputs of the relay board are connected to Raspberry Pi GPIO pins 2, 3, 4, and 17 respectively. Also, GND and +3.3V pins of the development board are connected to GND and VCC pins of the relay board. You must make sure that jumper JD is removed from the board. Connect an external +5V power supply to the JD-VCC pin of the relay board.

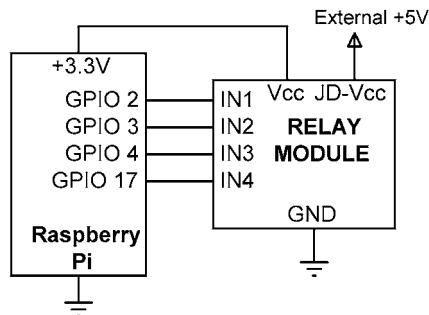


Figure 19.7 Circuit diagram of the project

Node-RED flow program: Figure 19.8 shows the flow program of the project which consists of 8 nodes: a **http in** node to get the HTTP request, a **template** node to store the HTML code, a **http out** node, a **function** node to format the data for the output, and 4 **rpi gpio out** nodes to control the 4 relay inputs.

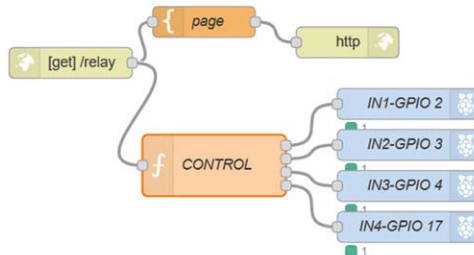


Figure 19.8 Flow program of the project

The steps are as follows:

- Create a **http in** node and configure as shown in Figure 19.9. Here, the URL is named as **/relay**. The operation of the project is as follows: When the user enters the URL **192.168.1.202:1880/relay** from a web browser, a GET request is sent to the Raspberry Pi node **http in**. This node then activates node **template** which displays a form on the user screen. This form contains buttons that the user can click to activate/de-activate a chosen relay. When a button is clicked, a command is received by node **http in**. This command is passed to node **function** which extracts the appropriate part of the command and outputs code to activate the correct **rpio gpio out** node, which in turn controls the relay connected to that node.



Figure 19.9 Configure the http in node

- Create a template node and enter the HTML code shown in Figure 19.20 into this node. At the beginning, the heading **Node-RED RELAY CONTROL** is displayed. The next line displays the sub-heading **Web Server Example With 4 Relays**. The remainder of the HTML code displays a form with the names of the relays as **RELAY1 ON** and **RELAY1 OFF** to **RELAY4 ON** and **RELAY4 OFF** where the ON states are displayed in green colour and the OFF states in red colour. The form elements are buttons that can be clicked. One line of the code is shown below. In this code, the button is named RELAY1 ON and it is green colour. Similar code is repeated for all the relays.

```
<button name="RELAY1" button style="background:green", value="ON" type="submit"><b>RELAY1 ON</b></button>
```

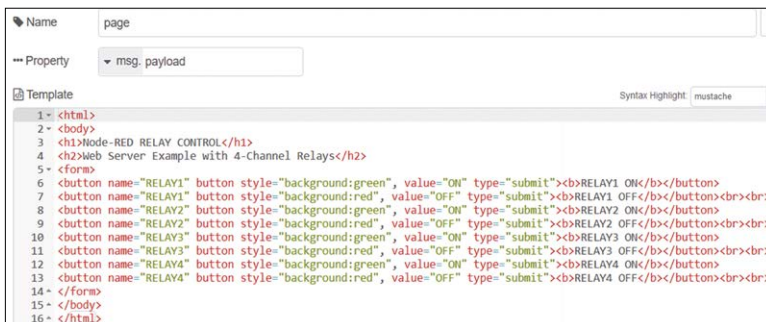


Figure 19.10 HTML code for the template node

When the GET request is received by node http in (i.e. when the user enters: 192.168.1.202:1880/relay), the form shown in Figure 19.21 is displayed on the user screen.

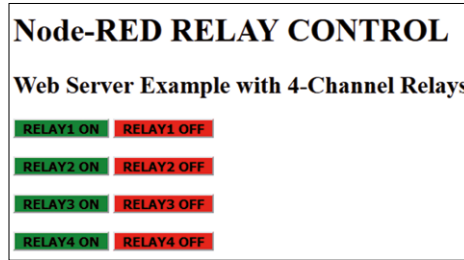


Figure 19.11 Form displayed on the user screen

- Create a **function** node named **CONTROL** having 4 outputs, and enter the following statements inside this node:

```
var var1 = null;
var var2 = null;
var var3 = null;
var var4 = null;

var T = msg.req.url;
switch (T)
{
    case "/relay?RELAY1=ON":
        var1 = {payload: 0};
        break;
    case "/relay?RELAY1=OFF":
        var1 = {payload: 1};
        break;
    case "/relay?RELAY2=ON":
        var2 = {payload: 0};
        break;
    case "/relay?RELAY2=OFF":
        var2 = {payload: 1};
        break;
    case "/relay?RELAY3=ON":
        var3 = {payload: 0};
        break;
    case "/relay?RELAY3=OFF":
        var3 = {payload: 1};
        break;
    case "/relay?RELAY4=ON":
        var4 = {payload: 0};
        break;
    case "/relay?RELAY4=OFF":
        var4 = {payload: 1};
        break;
}
return[var1, var2, var3, var4];
```

When a button is clicked on the form, a command is sent to the web server in addition to some other data. Here, we are only interested in the actual command sent. Variable `msg.req.url` is loaded with the command received by node `http` in when a button is clicked. The following are the commands received:

Button Pressed	Command sent to web server
RELAY1 ON	/relay?RELAY1=ON
RELAY1OFF	/relay?RELAY1=OFF
RELAY2 ON	/relay?RELAY2=ON
RELAY2 OFF	/relay?RELAY2=OFF
RELAY3 ON	/relay?RELAY3=ON
RELAY3OFF	/relay?RELAY3=OFF
RELAY4 ON	/relay?RELAY4=ON
RELAY4 OFF	/relay?RELAY4=OFF

The function uses a switch statement to check the message received. For example, if button **RELAY 1 ON** is clicked then `msg.req.url` will be loaded with the data `/relay?RELAY1=ON`. Variable `var1` is then loaded with 0 (remember that a relay is activated if its input is at logic 0) to activate Relay 1, connected to GPIO 2. Similarly, if **RELAY 1 OFF** is clicked then `msg.req.url` will be loaded with the data `/relay?RELAY1=OFF`. Variable `var1` is then loaded with 1 to de-activate Relay 1.

- Create 4 **gpio gpio** out nodes. Set the Pins of these nodes to GPIO 2, GPIO 3, GPIO 4, and GPIO 17, corresponding to relay inputs IN1, IN2, IN3, and IN4 respectively. Set the Types to digital output and Initialize pin states to High (1) so that all the outputs are at logic 1 when the program starts. This ensures that all the relays are de-activated at the beginning of the program.
- Join all nodes and click **Deploy**.

Testing

- Connect the relay module to the Raspberry Pi as shown in Figure 19.7.
- Open a web browser on your PC and enter: **192.168.1.202:1880/relay**
- You should see the form shown in Figure 19.21 displayed on your PC screen
- Click button RELAY1 ON. You should hear relay 1 activated, and the LED corresponding to this relay will turn ON on the relay module. Try turning ON/OFF other relays to make sure that they all work. Figure 19.22 shows the URL when relay 1 is activated.



Figure 19.12 Relay 1 is activated

- You can also control the relays from your smartphone. Enter the URL: **192.168.1.202:1880/relay** as shown in Figure 19.23 and you will be presented with the form to control the relays

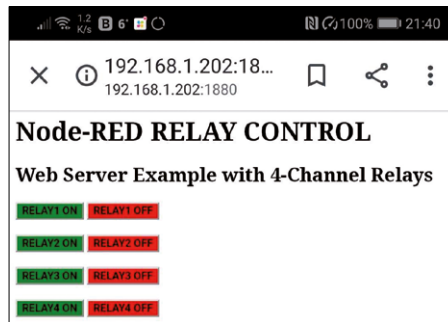


Figure 19.13 Controlling the relay from your smart phone

19.5 Summary

In this chapter, we have learned how to use the HTTP request node to request a web page. Also, an example project is given to show how a web server can be designed to control 4 relays from a web browser.

Appendix A • The function node

A.1 Overview

The function node is one of the most useful nodes in Node-RED and it is based on JavaScript (JS). In this Appendix, we will look at how to write statements inside functions, and also learn the very important topics of string manipulations, built-in string, and mathematical functions, conditional statements, repetition, etc that can be used in function nodes.

A.2 Variables

The nice thing about JS is that it does not care about the data types. Variables are declared using the **var** keyword. Every statement must be terminated with a semicolon. Some example variables are:

```
var name = "John Smith"; // string variable
var count = 10;          // integer variable
var cnt = "20";          // string variable
```

Variables in JS can be:

- Numeric e.g. integer and float
- String e.g. enclosed in " " or in ' '
- Boolean e.g. true or false
- Object e.g. JSON objects enclosed in { }
- Null e.g. empty or unassigned

An integer number has no decimal point such as $x = 25$. A floating point number has decimal point such as $x = 25.322$. Numbers can be written in scientific format as shown below:

```
x = 345e2            // x = 34500
or
x = 345e-2           // x = 3.45
```

All variables are destroyed except the 3 types of variables: global, context, and flow.

global variables: these variables retain their values across all flows. An example is given below where integer 10 is stored in variable XYZ, and then variable abc is assigned the value of XYZ:

```
global.set('XYZ', 10);
var abc = global.get('XYZ');
```

context variables: these variables retain their values in function nodes where they are declared only. An example is given below where variable cnt is initialized to 0 if it does not already exist. Then, it is incremented by one and stored back in cnt. Therefore, each time the function is run the value of cnt will be incremented by one:

```
var cnt = context.get('cnt') || 0;
cnt++;
context.set('cnt', cnt);
```

flow variables: these variables retain their values in the same flow. An example is given below where again integer 10 is stored in XYZ, and then variable abc is assigned the value of XYZ.

```
flow.set('XYZ', 10);
var abc = flow.get('XYZ');
```

The following basic mathematical operations can be performed in JS:

Valid arithmetic operators: +, -, *, /, **, %, ++, --
 Valid logical operators: <, >, ==, <=, !=
 Valid assignment operators: =, +=, *=, -=, /=, %=
 Valid comparison operators: ==, ===, !=, !==, >, <, >=, <=
 Valid logical operators are: &&, ||, !
 Valid bitwise operators are: &, |, ~, <<, >>, ^

Note the difference between the == and the === comparison operators. == is used to compare two variables where the type of the variable (e.g string or number) is not considered. On the other hand, === is used to compare two variables including their types. For example, if a = 10, then a == 10 is true and a == 5 is false. Also, a === 10 is true, but a === "10" is false.

A.3 Multiple outputs

It is sometimes necessary for a function to have more than one output. This can be done as shown in the following example:

```
var msg1 = { payload: "Message for output 1" };
var msg2 = { payload: "Message for output 2" };
return [ msg1, msg2];
```

A.4 String manipulation

String manipulation is one of the most important operations in JS. Examples of commonly used string manipulation operations are given in this section.

Number

Converts a string to a number.

```
var x = "10";           // string "10"
var y = Number(x);      // y is numeric 10
```

String

Converts a number to a string

```
var x = 10;           // x is number 10
var y = String(x);    // y is string "10"
```

Other string functions

Assuming that:

```
var x = "my name, is John, I am, 20 years old";
var y = "John Smith";
```

Then:

```
var L = y.length;           // y is the length of the string (L = 10
                             // in this case)

var pos = x.indexOf("name"); // first occurrence of name, starting from
                             // 0 (pos = 3). If the text is not found
                             // then -1 is returned

var pos = x.indexOf("name", 12); // look for name, starting from position
                                 // 12. pos = -1

var pos = x.search("name");    // search for name and return its position
                                 // (pos = 3)

var str = x.slice(3, 7);       // extract string starting from position 3
                             // and ending with position 7 (str =
                             // name). End position is not included
                             // (i.e. 3 to 6)

var str = x.slice(-8, -6);     // count from the end. Extract from
                             // position 6 to 8. str = ea

var str = y.slice(2);         // extract from 2 to the end (str = hn
                             // Smith)

var str = x.substr(3, 4);     // extract starting from position 3, and
                             // take 4 characters str = name

var str = y.substr(3);        // extract starting from position 3 to the
                             // end str = n Smith
```

```

var str = y.substr(-3);           // extract counting from the end.
                                   // str = ith

var r = y.replace("Smith", "Jones"); // replace string. r = John Jones.
                                   // Only the first occurrence is
                                   // replaced

var r = y.replace(/h/g, "s");     // replace all occurrences of h with
                                   // s. r = Josn Smits

var u = y.toUpperCase();           // change to upper case. u = JOHN SMITH

var s = y.toLowerCase();           // change to lower case. s = john smith

var c = x.concat(" ", y);         // join x and y.

                                   // c = my name, is John, I am, 20 years
                                   // old John Smith

var c = x + " " + y;              // join x and y.
                                   // c = my name, is John, I am, 20 years
                                   // old John Smith

var t = y.trim();                 // remove leading and trailing spaces.
                                   // t = John Smith

var r = y.charAt(0);              // return character at position. r = J

var s = x.split(",");             // split the string on commas
                                   // s[0] = my name
                                   // s[1] = is John
                                   // s[2] = I am
                                   // s[3] = 20 years old

var s = y.split("");              // split individual characters
                                   // s[0] = J
                                   // s[1] = o
                                   // s[2] = h
                                   // s[3] = n
                                   // s[4] =
                                   // s[5] = S
                                   // s[6] = m
                                   // s[7] = i
                                   // s[8] = t
                                   // s[9] = h

```

```
var s = y.repeat(2);           // repeat string y twice.
                                // s = John Smith John Smith

var n = y.endsWith("Smith");   // check if string y ends with Smith.
                                // n becomes true

var n = y.startsWith("Smith"); // check if string y starts with Smith.
                                // n becomes false

var n = y.includes("John");     // check if string y includes "John".
                                // n becomes true
```

A.5 Mathematical functions

The following mathematical functions are commonly used. All of these functions must be preceded by the keyword **Math**. For example, **Math.abs(x)**:

```
var x = sqrt(64);              // square root. x = 8

var p = PI;                    // p = 3.14159

var s = sin(r);                // trigonometric sine, r in radians

var c = cos(r);                // trigonometric cosine, r in radians

var t = tan(r);                // trigonometric tangent, r in radians
```

To convert degrees to radians, multiply by Pi and divide by 180. For example, to calculate sin 30 degrees:

```
var s = Math.sin(30*Math.PI/180.0); // s = 0.5

var x = abs(y);                 // absolute value of y

var x = ceil(y);               // y rounded upwards to nearest integer

var x = floor(y);              // y rounded downwards to nearest integer

var y = exp(x);                // exponentiation of x, y=ex

var r = pow(x, y);             // r = xy

var y = round(x);              // round x to the nearest integer

var y = trunc(x);              // return integer part of x

var y = log(x);                // y = Lne(x)
```

```

var y = LN10(x);           // y =  $\ln_{10}(x)$ 

var y = asin(x);           // returns arcsine of x in radians

var y = acos(x);           // returns arccosine of x in radians

var y = atan(x);           // returns arctangent of x

var r = atan2(x, y);       // returns arctangent of y/x

```

Hyperbolic functions such as `sinh`, `cosh`, `tanh`, `asinh`, `acosh`, and `atanh` are also supported.

A.6 Number conversions and checking numbers

The following number manipulations can be made:

Assuming `x = 123.47`, `z = 25`, and `q = "45"`

```

Number.isInteger(x);       // if x is an integer number. Returns false

var y = x.toFixed(1);       // 1 digits after decimal point. y = 123.4

var y = x.toFixed(4);       // 4 digits after decimal point. y = 123.4700

var y = x.toPrecision(3);   // Format to length 2. y = 123

var y = x.toPrecision(4);   // y = 123.5

var y = x.toPrecision(7);   // y = 123.4700

var s = z.toString();       // convert number to string. s = "25"

var s = z.toString(2);      // convert with base 2. s = 11001

var s = z.toString(16);     // convert with base 16. s = 10

var n = parseInt(q);        // parse string and return integer. n = 45
                             // spaces are ignored and only the first number
                             // is returned. A radix can be specified

var n = parseFloat(q);      // parse string and return float. n = 45
                             // spaces are ignored and only the first number
                             // is returned, A radix can be specified

var n = eval("2 * z");      // evaluate expression. n = 50

```

A.7 Date

Date functions are useful when we want to include the date and/or time in our function nodes. Some of the commonly used date functions are given in this section. Data objects are created with `new Date()`. Some commonly used functions are:

```
var dt = new Date();           // get date. e.g. Thu Dec 26 2019 10:52:02
                                // GMT+0000 (Greenwich Mean Time)

var d = dt.getDate();         // day of the month. eg. d = 26

var d = dt.getDay();          // day of the week (Monday = 1). e.g. d = 4

var d = dt.getFullYear();     // year, e.g. d = 2019

var d = dt.getHours();        // hours, e.g. d = 22

var d = dt.getMinutes();      // minutes, e.g. d = 40

var d = dt.getSeconds();      // seconds, e.g. d = 20

var d = dt.toString();        // today's date as string, d = Thu Dec 26
                                // 2019
```

Additionally, the date and time can be set using functions: `setHours()`, `setMinutes()`, `setSeconds()` etc.

A.8 Arrays

Arrays are very useful in all programming languages as it enables related items to be collected and accessed under a name. In JS arrays are indexed starting from 0.

Assuming that we have the array declaration:

```
var cities = ["London", "Paris", "Ankara"];
var counties = ["UK", "France", "Turkey"];
```

We can use the following functions (only the commonly used functions are given here):

```
var all = cities.concat(counties); // join arrays. all = ["London",
                                // "Paris", "Ankara", "UK", "France",
                                // "Turkey"]

var all = Array.from("xyz");       // create array from string. all =
                                // ["x", "y", "z"]

var n = cities.includes("Paris");  // check if array cities includes
                                // Paris. n becomes true
```



```
var n = cities.indexOf("Paris");    // return position of Paris in cities.
                                    // n   = 1. If the string is not found
                                    // then -1 is returned

var s = cities.join();              // join the array as a string separated
                                    // with commas. s =
                                    // "London,Paris,Ankara"

var n = cities.length;              // length of array cities. n = 3

var s = cities.pop();               // remove last element of array cities
                                    // and then return it. s = "Ankara"

var s = cities.shift();              // remove first element of array cities
                                    // and then return it. s = "London"

var s = cities.push("Cairo");        // add a new element Cairo to the end
                                    // of array cities and returns the new
                                    // length of cities

var s = cities.reverse();            // reverse the elements of array
                                    // cities. s = ["Ankara", "Paris",
                                    // "London"]

var s = cities.sort();               // sort array cities. s = ["Ankara",
                                    // "London", "Paris"]

var s = cities.slice(1, 2);          // return selected elements of array
                                    // cities, starting from position 1 to
                                    // 2. s = [ "Paris" ]
```

A.9 Conditional statements

Conditional statements are used when a choice is to be made in a program. JS supports the following conditional statements:

if..else..elseif

This is one of the commonly used conditional statements. An example is given below. If multiple statements are used inside a condition they must be enclosed in a curly bracket:

```
if(x > 10)
{
    a= 0;
    b = 0;
}
else if(x < 10)
{
    a = 1
```

```
    b = 1;
  }
  else
    c = 0;
```

switch

The switch statement is used to select one of many conditions. An example is given below:

```
switch(n)
{
  case 1:
    text = "you selected 1";
    break;
  case 2:
    text = "you selected 2";
    break;
  case 3:
    text = "you selected 3";
    break;
  default:
    text = "you selected something else";
}
```

A.10 Repetition (loops)

Repetition statements are used when it is required to create loops in programs. JS supports the following repetition statements:

do..while

an example is given below where the loop is repeated 5 times. At the end of the loop, k = 5

```
var k = 0;
do
{
  k++;
}while(k < 5);
```

for

repeat the loop a number of times. In the following example the loop is repeated 5 times and the value of j = 5 at the end of the loop:

```
var k;
var j = 0;
for(k = 0; k < 5; k++)
{
  j++;
}
```

while

Loop through while a condition is true. In the following example, the loop is repeated while $k < 5$. At the end of the loop, $k = 5$

```
var k = 0;
while (k < 5)
{
    k++;
}
```

for..of

Loop through the values of an array. An example is given below. At the end of the loop, variable $y = \text{"LondonParisAnkara"}$:

```
var cities = ["London", "Paris", "Ankara"];
var k;
var y = "";

for (k of cities)
{
    y = y + k;
}
```

break

Used to exit from a loop when a condition becomes true. In the following example, the loop terminates when $k = 3$. At the end of the loop $j = 4$

```
var k;
var j = 0;
for(k = 0; k < 10; k++)
{
    j++;
    if(k == 3)
        break;
}
```

continue

Skip the loop when a condition becomes true. In the following example, the sum of numbers from 0 to 10 are calculated. When $k = 5$, the loop continues but misses this value. At the end of the loop, $j = 50$ and not 55

```
var k;
var j = 0;
for(k = 0; k <= 10; k++)
{
    if(k == 5)continue;
}
```

```
j = j + k;
}
```

A.11 Functions

Functions are useful when a certain operation is to be repeated several times in a program. Functions are also used to breakdown a complex program into smaller more manageable sections. In JS a function is declared with the keyword `function`, followed by the name of the function. A function can have optional parameters inside brackets. The body of a function must be enclosed inside curly brackets.

In the following example, a function called `Add` is created which has 2 arguments. The function adds these arguments and returns the result to the calling program:

```
function Add(a, b)
{
    return a + b;
}
```

Call the function as follows:

```
y = Add(3, 4)
```

Where `y` takes the value of 7

A.12 Examples

Some examples are given below to show how some of the built-in functions can be used in function nodes.

Example

Figure A.1 shows a flow diagram with a function node and a debug node. Assuming the input to the function node is:

```
T = 15, 500, 752
```

Write a function with 3 outputs to extract 15, 50, and 75. Display the results in the Debug window.

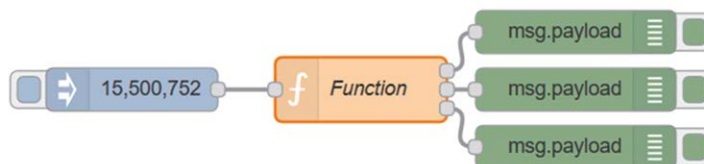


Figure A.1 Flow program of Example 1

Solution

The function should have the following statements. `split()` function is used to extract the elements:

```
var F = msg.payload.split(",");  
var1 = {payload: F[0]};  
var2 = {payload: F[1]};  
var3 = {payload: F[2]};  
return [var1, var2, var3];
```

Figure A.2 shows the data displayed in the Debug window.



Figure A.2 Data displayed in the Debug window

Appendix B • Flow programs used in the book

A.1 Overview

The web site of the book includes all the flow programs used in this book. The programs are stored as word documents and are given the name of the Project that they belong to, e.g. Project 29. Some flow programs do not belong to a project and for these, the name of the figure they belong to is given. e.g. Figure 12.14

A.2 Using the flow programs

The steps to use the flow programs from the book web site are as follows:

- Select the required flow program from the web site
- Open the program and copy it to the Clipboard
- Start Node-RED Dashboard and click **Menu -> Import**
- Paste the program to the opened window and click **Import**
- You should see the flow program added to the Node-RED workspace
- Click **Deploy**

Appendix C • Components used in the book

Processors:

Raspberry Pi 4, ESP32 DevKitC, ESP8266 NodeMCU, Arduino Uno

Components:

- 4 x Red LED
- 4 x 470 Ohm resistor
- 1 x Green LED
- 1 x Orange LED
- 1 x 10K pot
- 1 x 220 Ohm resistor
- 2 x 1K resistors
- 2 x 2K resistors
- 1 x Buzzer
- 1 x Button
- 1 x I2C LCD
- 1 x Parallel LCD
- 1 x DHT11
- 2 x Ultrasonic sensors
- 1 x MCP3002 ADC
- 1 x TMP36DZ sensor
- 1 x GPS Click board
- 2 x tRF Click Board
- 2 x Antenna for tRF
- 1 x Sense HAT
- 1 x 4 Relay module (Elegco)

Index**A**

Acceleration, 230
 ADC, 147
 Advanced nodes, 58
 Alexa, 263
 Amazon Alexa, 263
 Analog sensor, 146
 Analysis nodes, 58
 Arduino, 199, 245
 Arrays, 315
 Atmospheric pressure, 76
 Auto-pan, 213

B

Bar chart, 157
 Baud rate, 198
 Bluetooth, 272
 Break, 318
 Broker, 279
 Button, 106
 Button node, 163
 Buzzer, 92, 135

C

Cat, 34
 Chart, 156
 Chat, 184
 Comment node, 106
 Context variable, 87, 309
 Compass heading, 229
 Conditional statements, 316
 Continue, 318
 Copying node, 55
 Core nodes, 55
 CSI port, 16

D

Dashboard, 153
 Dashboard nodes, 153
 Date and time, 47
 DELETE, 300
 DevKitC, 257
 DHT11, 124, 253
 DHT22, 124

Disk usage, 39
 Distance sensor, 129
 DSI port, 16
 Duty cycle, 111

E

Email, 74, 93
 Email in, 93
 Email out, 97
 ESP32, 257
 ESP8266, 283
 Etcher, 21
 Event counter, 122
 Exporting, 54

F

File operation, 33
 File permission, 28
 Firmata, 247
 Flashing LED, 85
 Flow variables, 309
 Functions, 319
 Function nodes, 57, 309

G

Gauge, 154, 251
 GET, 300
 GPIO connector, 80
 GPIO pins, 81
 GPS, 204
 GPS click, 206
 Gyroscope, 227

H

Halt, 40
 HC-SR04, 129
 HD44780, 137
 HDMI, 16
 Horizontal bar chart, 158
 HTTP, 300
 HTTP request, 57
 Humidity, 120

I

Ifconfig, 42
 Importing, 54

Input nodes,	55	P	
ISM,	222	Parallel LCD,	137
J		Ping,	187
Jason,	72	Pitch,	232
Joystick,	233	PoE port,	16
L		Pressure,	242
Latitude,	208	Ps,	39
LCD,	113	POST,	300
LCD character set,	148	Publisher,	279
LED colours,	235	Push button,	106
LED matrix,	234	PUT,	300
Line chart,	156	Putty,	23
LM35DZ,	201	PWM,	110
Logic level converter,	217	R	
Longitude,	208	Random node,	121
M		Random number,	60
Map,	212	Raspberry Pi nodes,	58
Mark,	110	Raspbian Buster,	18
Mathematical functions,	313	Remote access,	22
MCP3002,	142	Repetition,	317
Mean,	67	RF radio,	222
Micro SD card,	17	Roll,	232
MOSFET,	216	Rotate screen,	243
Mosquitto,	280	S	
Motion events,	229	SCL,	113
MQTT,	278	Scrolling data,	242
MQTT in,	281	SDA,	113
MQTT out,	281	Send UDP,	178
Multiple GPIO,	98	Sense Hat,	225
Multiple gauge,	160	Sense Hat nodes,	226
Multiple inputs,	62	Serial communication,	198
Multiple LED,	173	Shutdown,	40
Multiple outputs,	65	Slider node,	161
N		Smooth node,	67
Ngrok,	269	Social nodes,	58
NodeMCU,	285	Space,	110
Node-RED configuration,	78	SPI,	143
O		SSH,	23
Openweathermap,	71, 91	Storage nodes,	58, 190
Orientation,	231	String manipulation,	310
Output nodes,	56	Subscriber,	279
		Switch node,	109
		Super user,	37
		Switch,	317

Switch node, 165

T

Talking weather forecast, 166

TCP, 170

Temperature conversion, 52

Text node, 165

TightVNC, 26

Timestamp, 190

TMP36, 146

Topics, 279

Traffic lights, 102

Trf click, 223

Trigger node, 86. 104

Twitter, 76

Twitter in, 76

Twitter out, 76

U

UDP, 170

UDP receive, 176

Ultrasonic, 129

Unit conversion, 62

URL, 301

USB keyboard, 17

USB port, 220

Utf-8, 195

V

Variables, 309

VNC, 25

VNC server, 25

Voltmeter, 144

W

Weather, 70

Web server, 301

Websocket, 56

While, 318

Whoami, 36

WiFi, 170

Wildcard, 33

Y

Yaw, 232

Programming with Node-RED

Dogan Ibrahim



Prof. Dr. Dogan Ibrahim is a Fellow of the Institution of Electrical Engineers. He is the author of over 60 technical books, published by publishers including Wiley, Butterworth, and Newnes. He is the author of over 250 technical papers, published in journals, and presented in seminars and conferences.

The Internet of Things (IoT) is becoming a major application area for embedded systems. As a result, more and more people are becoming interested in learning about embedded design and programming. Technical colleges and universities are moving away from legacy 8 and 16-bit microcontrollers and are introducing 32-bit embedded microcontrollers to their curriculums. Many IoT applications demand precision, high processing power, and low power consumption.

Produced by IBM, Node-RED is an open-source visual editor for wiring the Internet of Things. Node-RED comes with a large number of nodes to handle a multitude of tasks. The required nodes are selected and joined together to perform a particular task. Node-RED is based on flow type programming where nodes are configured and joined together to form an application program. There are nodes for performing complex tasks, including web access, Twitter, E-mail, HTTP, Bluetooth, MQTT, controlling GPIO ports, etc. One particularly nice aspect of NodeRED is that the programmer does not need to learn how to write complex programs. For example, an email can be sent by simply joining nodes together and writing only a few lines of code.

The aim of this book is to teach how Node-RED can be used in projects. The main hardware platform used with most of the projects in this book is Raspberry Pi 4. Chapters are included to show how Node-RED can also be used with Arduino Uno, ESP32 DevKitC, and the ESP8266 NodeMCU microcontroller development boards.

ISBN 978-1-907920-88-2



Elektor International Media BV
www.elektor.com