Pradeeka Seneviratne, John Sirach

Raspberry Pi 3 Projects for Java Programmers

Get the most out of your Raspberry Pi 3 with Java

Packt

Raspberry Pi 3 Projects for Java Programmers

Get the most out of your Raspberry Pi 3 with Java

Pradeeka Seneviratne
John Sirach

BIRMINGHAM - MUMBAI

Raspberry Pi 3 Projects for Java Programmers

First published: May 2017

Production reference: 1300517

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.
ISBN 978-1-78646-212-1

www.packtpub.com

Credits

Authors
Pradeeka Seneviratne
John Sirach

Reviewer
Rajdeep Chandra

Commissioning Editor
Pratik Shah

Acquisition Editor
Prachi Bisht

Content Development Editor
Trusha Shriyan

Technical Editor
Varsha Shivhare

Copy Editor
Safis Editing

Project Coordinator
Kinjal Bari

Proofreader
Safis Editing

Indexer
Rekha Nair

Graphics
Kirk D'Penha

Production Coordinator
Shantanu Zagade

About the Authors

Pradeeka Seneviratne is a software engineer with over 10 years of experience in computer programming and designing systems. He is an expert in the development of Arduino and Raspberry Pi-based embedded systems.

Pradeeka is currently a full-time embedded software engineer who works with embedded systems and highly scalable technologies. Previously, he worked as a software engineer for several IT infrastructure and technology servicing companies.

He collaborated on the Outernet (free data from space, forever) project as a volunteer hardware and software tester for Lighthouse, and Raspberry Pi-based DIY Outernet receivers based on Ku band satellite frequencies.

He is also the author of three books:

- *Internet of Things with Arduino Blueprints* by Packt Publishing
- *IoT: Building Arduino-Based Projects* by Packt Publishing
- *Building Arduino PLCs* by Apress

John Sirach works as a product owner at Greenhouse Innovation. He has more than 10 years of experience in Internet-related disciplines from connectivity to hosting, and Internet of Things. Currently, he is involved in the open source PiDome home automation platform project as a passionate Java and JavaFX software developer and project maintainer.

In the past ten years, he has gained experience with large-scale web applications committed to online services with most experience gained in frontend web development and application middleware.

www.PacktPub.com

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com`and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www.packtpub.com/mapt`

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at `https://www.amazon.com/dp/1786462125`.

If you'd like to join our team of regular reviewers, you can e-mail us at `customerreviews@packtpub.com`. We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!

Table of Contents

Preface

As Java becomes widely used on different hardware platforms from computers to embedded devices, Raspberry Pi has no limitations to work with it to gain full power of Java. This book presents some basic to advanced projects that can be used to build Raspberry Pi 3 projects with Java as the core development platform.

What this book covers

`Chapter 1`, *Setting up Your Raspberry Pi*, teaches you about the hardware available with Raspberry Pi and how to prepare to utilize it from the installed Java Virtual Machine. Setting up the NetBeans editor to be able to write Java applications, which can be deployed from the editor, including the in-editor, available console to be able to interact with Java applications on Raspberry Pi.

`Chapter 2`, *Automatic Light Switch Using Presence Detection*, explains how to perform analogue readings using digital pins, as the Raspberry Pi has no analog reading capabilities. By adding a 16x2 character display, you will visualize these readings. By adding logic to the code, you will be able to determine when someone is present, and in combination with detecting light levels, you will be able to set a relay state that turns a light on or off.

`Chapter 3`, *A Social and Personal Digital Photo Frame*, offers comprehensive guidance on building a social and personal digital photo frame with the Raspberry Pi as the heart. Flickr will be used the social media platform to grab a set of images that will be shown, on the display. FEH will be used an X11 image viewer to display images as a slide show with customizable configurations. Some advanced configurations for the Raspberry Pi will be needed to automate the photo frame to connect with Flickr, download images to local storage, display them on screen, and frequently sync local storage with Flickr.

`Chapter 4`, *Integrating a Real-Time IoT Dashboard*, presents a real-time IoT dashboard to display sensor data and Raspberry Pi's system information on it, and control actuators from it. The dashboard will be built with Adafruit IO, in conjunction with the Eclipse Paho Java `MQTT` library and `Pi4J`. The IC2 communication protocol will be used to read data from the sensors by enabling the 12C interface on Raspberry Pi.

`Chapter 5`, *Wireless Controlled Robot*, introduces the Zumo chassis kit and how to build a Raspberry Pi brain on it with Java and Pi4J. The robot uses two gear motors to rotate the drive wheels, and the `Pi4J` library allows us to build various moving and turning mechanisms to control the robot. The built-in Wi-Fi module of the Raspberry Pi 3 allows you to connect the robot wirelessly to the computer, in order to execute the commands through SSH with a keyboard.

`Chapter 6`, *Building a Multipurpose IoT Controller*, teaches you how to build a simple web-based IoT controller by installing and configuring a Jetty servlet engine on the Raspberry Pi. Pi4J is used to control an LED or any other actuator from the web interface through a local network or from the Internet, by configuring port forwarding on the router.

`Chapter 7`, *Security Camera with Face Recognition*, explains how to build a security camera with face recognition using the `OpenCV` library for Java and the Raspberry Pi camera module. It uses built-in cascade classifiers to detect human faces and highlights them with a square in real-time video.

What you need for this book

The following software and hardware components are essential in order to build all the projects that presented in this book:

- Raspberry Pi 3
- Micro SD card
- HDMI screen or HDMI enabled TV
- HDMI to HDMI cable
- 5V power supply
- Breadboard and wires
- Raspberry Pi 3 T-Cobbler or male to female jumper wires
- Kohm resistors
- Hitachi HD44780 16X2 LCD
- A device capable of being detected by Bluetooth, such as a mobile phone
- Phillips head screw driver
- One Zumo Chassis kit (no motors)
- Two 100:1 micro metal Gearmotor HP 6V
- One H-bridge motor driver - SN754410 breakout board
- Aluminum Standoff
- Machine screws

- Two 0.1μF ceramic capacitors
- Four rechargeable NiMH AA batteries
- One USB battery pack for Raspberry Pi - 10000 mAh - 2 x 5V outputs
- 3mm LED
- Raspberry Pi camera module with mount
- Hookup wires

Who this book is for

This book is suitable for those who have experience in Java application development and are interested in developing applications on the Raspberry Pi development environment. A basic level of Java-based development skills is essential to develop the projects that will be discussed in this book.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Let's take a look at the `main(String args[])` method."

A block of code is set as follows:

```
/**
 * Run the LCD example.
 */
public final void runExample(){
 /// Clear the display
 handler.clear();
 /// Cursor to home position (0,0)
 handler.setHome();
 /// Write to the first line.
 handler.write("-- RASPI3JAVA --");
 /// Create a time format for output
 SimpleDateFormat formatter = new SimpleDateFor-
 mat("HH:mm:ss");
 /// Sets the cursor on the second line at the first posi-
     tion.
 handler.setCursor(1, 0);
 /// Write the current time in the set format.
```

```
    handler.write("--- " + formatter.format(new Date()) + " ---");
  }
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "In the top of the window there are a set of buttons present which when started has the **Welcome** button selected. On the same level there is a **Breadboard** button."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support`and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Raspberry-Pi-3-Projects-for-Java-Programmers`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `https://www.packtpub.com/sites/default/files/downloads/RaspberryPi3ProjectsforJavaProgrammers_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the **Errata** section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Setting up Your Raspberry Pi

In the first part of this chapter, we will be setting up a Raspberry Pi 3 with the installation of the official, by the Raspberry Pi foundation, released Raspbian Linux distribution with the help of the NOOBS installer. This will be followed up with any preparations needed to be able to use the onboard hardware and the installation of Java. The second part will cover preparing NetBeans as the editor of choice to write, compile, and deploy our Java applications with a simple Hello World application at the end of the chapter to confirm that our setup works.

Getting started with the Raspberry Pi

With the release of the Raspberry Pi 3, the Raspberry Pi foundation has made a very big step in the history of the Raspberry Pi. The current hardware architecture is now based on a 1.2 GHz 64 bit ARMv7. This latest release of the Raspberry Pi also includes support for wireless networking and has an onboard Bluetooth 4.1 chip available.

Even though the Raspberry Pi 3 has 64-bit ARMv8, the operating system of the Raspberry Pi which is Raspbian confusingly report it is an 32-bit ARMv7. This is because of the Raspbian OS is currently only available in 32-bit.

To get started with the Raspberry Pi you will need the following components:

- **Keyboard and mouse:** Having both a keyboard and mouse present will greatly help with the installation of the Raspbian distribution. Almost any keyboard or mouse will work.
- **Display:** You can attach any compatible HDMI display, which can be a computer display or a television. The Raspberry Pi also has composite output shared with the audio connector. You will need an A/V cable if you want to use this output.

- **Power adapter:** Because of all the enhancements made, the Raspberry Pi foundation recommends a 5V adapter capable of delivering 2.5 A. You would be able to use a lower-rated one, but I strongly advise against this if you are planning to use all the available USB ports. The connector for powering the device uses a micro USB cable.
- **MicroSD card**: The Raspberry Pi 3 uses a MicroSD card. I would advise using at least an 8-GB class 10 version. This will allow us to use the additional space to install applications, and as our projects will log data, you won't be running out of space soon.
- **The Raspberry Pi 3:** Last but not least, a Raspberry Pi 3. Some of our projects will be using the on-board Bluetooth chip, and this version will also be focused on in this book.

Our first step will be preparing a SD card for use with the Raspberry Pi. You will need a MicroSD card as the Raspberry Pi 3 only supports this format. The preparation of the SD card is being done on a normal PC, so it is wise to purchase one with an adapter fitting a full-size SD card slot. There are webshops selling preformatted SD cards with the NOOBS installer already present on the card. If you have bought one of these preformatted cards you can skip to the *Installing Raspbian* section.

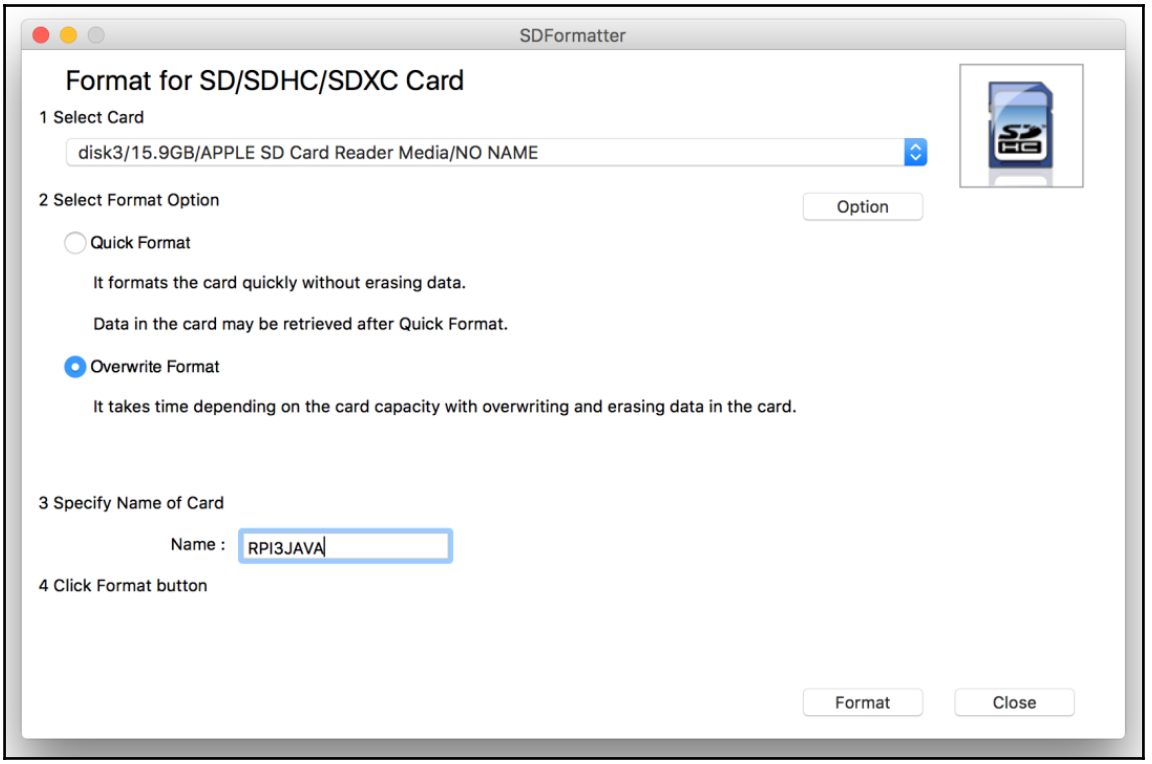
Getting a compatible SD card

There is a large number of SD cards available. The Raspberry Pi foundation advises an 8-GB card, which leaves space to install different kinds of application and supplies enough space for us to write log data. When you buy an SD card, it is wise to keep your eyes open for the quality of these cards. Well-known and established manufacturers often supply better quality than the counterfeit ones. SD cards are sold with different class definitions. These classes explain the minimal combined read and write speeds. Class `6` should provide at least 6 MB/s, and class `10` cards should provide at least 10 MB/s. There is a good online resource available that provides tested results of using SD cards with the Raspberry Pi. If you need a resource to check for compatible SD cards, I advise you to go to the embedded Linux page at `http://elinux.org/RPi_SD_cards`.

Preparing and formatting the SD card

To be able to use the SD card, it first needs to be formatted. Most cards are already formatted with the FAT32 filesystem, which the Raspberry Pi NOOBS installer requires, unless you have bought a large SD card it is possible it is formatted with the exFAT filesystem. These then should also be formatted as FAT32. To format the SD card, we will be using the SD association's SDFormatter utility, which you can download from `http://elinux.org/RPi_SD_cards`, as default OS supplied formatters do not always provide optimal results.

In the following screenshot, the SDFormatter for the Mac is shown. This utility is also available for Windows and has the same options. If you are using Linux, you can use GParted. Make sure when using GParted you use **FAT32** as the formatting option. As shown in the screenshot, select the **Overwrite format** option and give the SD card a label. The example shows **RPI3JAVA**, but this can be the label of your choice so you can quickly recognize the card when it's inserted:

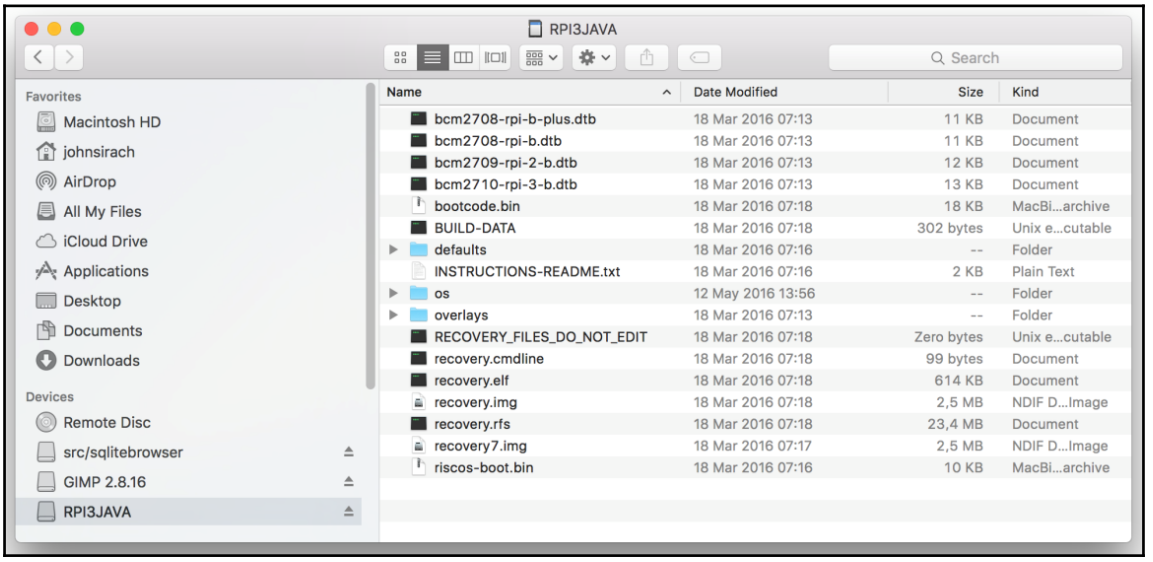


Press the **Format** button to start formatting the SD card. Depending on the size of the SD card, this could take some time, enabling you to get a cup of coffee. The utility will show a done message in the form of `Card Format complete` when the formatting is done. You will now have a usable SD card.

To be able to use the NOOBS installer, you will need to follow the following steps:

1. Download the NOOBS installer from `https://www.raspberrypi.org/downloads/`.
2. Unzip the file with your favorite unzip utility. Most OSes already have one installed.
3. Copy the contents of the unzipped file into the SD card's `root` directory so that the copy result is as shown in the following screenshot:

When selecting the NOOBS for download, only select the lite version if you do not mind installing Raspbian using the Raspberry Pi's network connection.



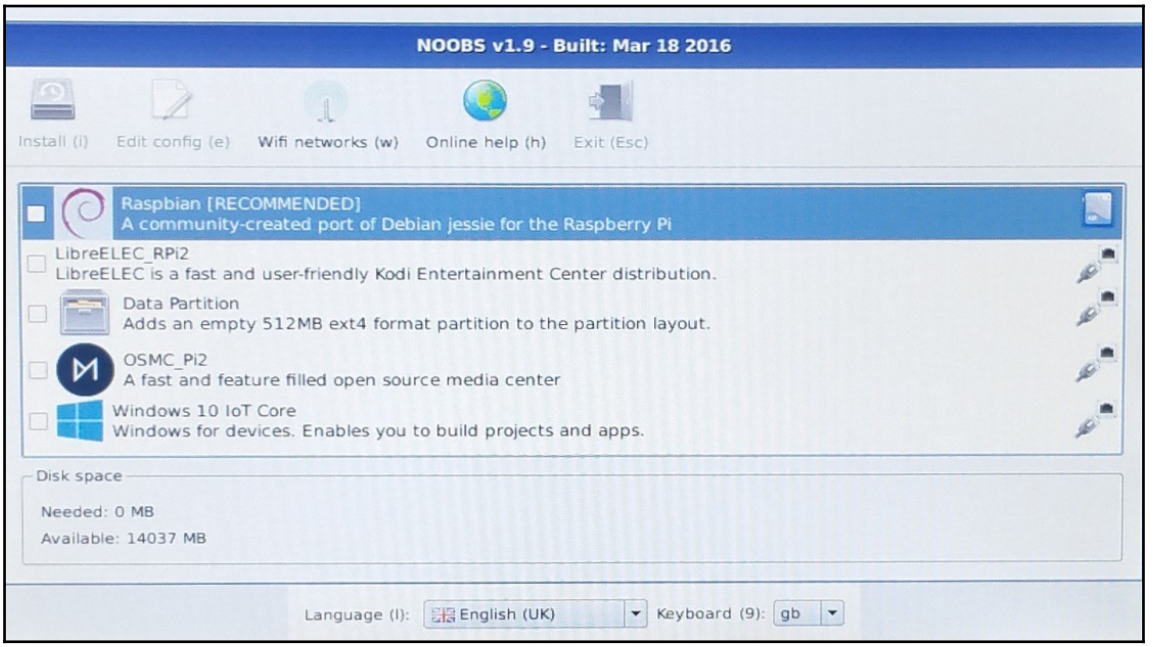
After we have copied the required files into the SD card, we can start installing the Raspbian OS.

Installing Raspbian

To install Raspbian, we need to get the Raspberry Pi ready for use. As the Raspberry Pi has no power **on** and **off** button, the powering of the Raspberry Pi will be done as the last step:

1. At the bottom of the Raspberry Pi on the side you will see a slot to insert your MicroSD card. Insert the SD card with the connectors pointing to the board.
2. Next, connect the HDMI or the composite connector and your keyboard and mouse. You won't need a network cable as we will be using the wireless functionality built into the Raspberry Pi.
3. We will now connect the Raspberry Pi with the micro USB power supply.
4. When the Raspberry Pi boots up, you will be presented with the OSes available to be installed. Depending on the download of NOOBS you have done, you will be able to see if the Raspbian OS is already available on the SD card or if it will be installed by downloading it. This is visualized by showing an SD card image or a network image behind the OS name. In the following screenshot, you see the NOOBS installer with the Raspbian image available on the SD card.
5. At the bottom of the installation screen you will find the **Language** and **Keyboard** drop-down menus. Make sure you select the appropriate language and keyboard selection; otherwise, it will be difficult to enter the right characters on the command line and in other tools requiring text input.

6. Select the **Raspbian [RECOMMENDED]** option and click the **Install (i)** button to start installing the OS:



7. You will be prompted with a popup confirming the installation as it will overwrite any existing installed OSes. As we are using a clean SD card, we will not be overwriting any.

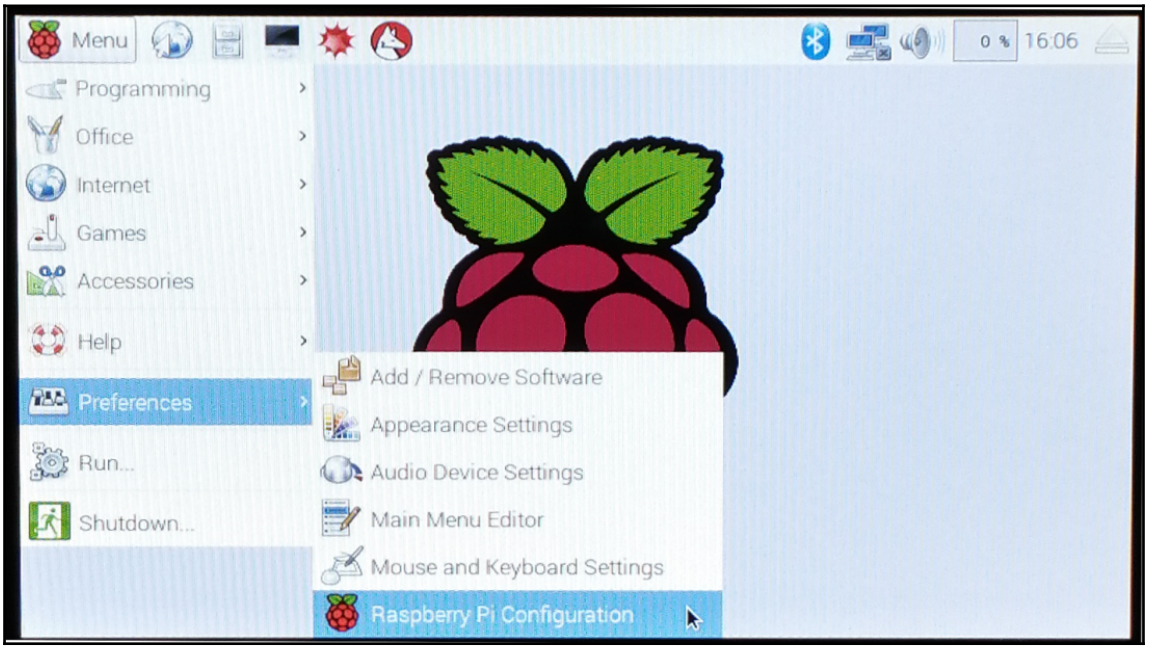
It is safe to press Yes to start the installation. This installation will take a couple of minutes, so it is a good time to go for a second cup of coffee.

8. When the installation is done you can press **Ok** in the popup that appears and the Raspberry Pi will reboot. Because Raspbian is a Linux OS, you will see text scrolling by that relates to services that are being started by the OS.
9. When all services are started, the Raspberry Pi will start the default graphical environment, called LXDE, which is one of the Linux window managers.

Configuring Raspbian

Now that we have installed Raspbian and have it booting into the graphical environment we can start configuring the Raspberry Pi for our purposes. To be able to configure the Raspberry Pi, the graphical environment has a utility tool installed, which eases up the configuration called Raspberry Pi Configuration:

1. To open this tool, use the mouse and click on the **Menu** button on the top left, navigate to **Preferences**, and press the **Raspberry Pi Configuration** menu option, as shown in the following screenshot:



2. When you click on the **Raspberry Pi Configuration** tool **Menu** option a popup will appear with the graphical version of the `raspi-config` command-line tool. In the graphical popup we see four tabs explaining different parts of possible configuration options. We first focus on the System tab, which allows us to:
    - Change the **Password**
    - Change the **Hostname**, which helps to identify the Raspberry Pi in the network
    - Change the `Boot` method to **To Desktop** or the **CLI**
    - Set the **Network at Boot** option

3. With the system newly installed, the default username is **pi** and the password is set to `raspberry`. Because these are the default settings, it is recommended that we change the password.
4. Press the **Change Password** button and enter a newly chosen password twice, Once to set the password and the second time to make sure we have entered the new password correctly.
5. Press **Ok** when the password has been entered twice.

Try to come up with a password that contains capital letters, numbers, and some strange characters, as this will make it more difficult to guess.

6. Now after we have set a new password, we are going to change the hostname of the Raspberry Pi. With the hostname we are able to identify the device on the network. I have changed the hostname into `RASPI3JAVA`, which helps me to identify the Raspberry Pi to be used for the book.

The hostname is used on the CLI, so you will immediately identify this Raspberry Pi when you log in.

7. By default, the Raspberry Pi boots into the graphical user interface, which you are looking at right now. Because a future project will require us to make use of a display with our application, we will be choosing to boot into CLI mode.
8. Click on the **radio** button that says **To CLI**. The next time we reboot, we will be shown the CLI.
9. Because we will be using the integrated Wi-Fi connection on this Raspberry Pi, we are going to change the **Network at Boot** option to have it waiting for the network. Later on in the book we will be creating network-based services and we will be logging in from remote locations. Having the network ready before any service requiring a network connection is a good practice.
10. Tick the box that says **Wait for network**.

We have set some default settings, which sets some primary options to help us identify the Raspberry Pi and change the boot mode. We will now be changing some advanced settings that enable us to make use of the hardware provided by the Raspberry Pi:

1. Click on the **Interface** tab, which will give us a list of the available hardware provided. This list is as follows:
    - **Camera:** The Official Raspberry Pi camera interface
    - **SSH:** To be able to log in from remote locations
    - **SPI:** Serial peripheral interface bus for communicating with hardware
    - **I2C:** Serial communication bus mostly used between chips
    - **Serial:** The serial communication interface
    - **1-Wire:** Low data power supplying bus interface conceptual based on I2C
    - **Remote GPIO**

2. A future project that we will be working on will require some kind of camera interface. This project will be able to use both the local attached official Raspberry Pi Camera module as well as a USB connected webcam.
3. If you have the camera module, tick the **Enabled** radio box behind **Camera**. We will be deploying our applications immediately from the editor. This means we need to enable the **SSH** option. By default this already is, so we leave the setting as it is. If the **SSH** option is not enabled, tick the **Enabled radio** button behind the **SSH** option. For now, you can leave the other interfaces disabled as we will only enable them when we need them.
4. As we now have enabled the default interfaces that we will need, we are going to do some performance tweaking. Click on the **Performance** tab to open the performance options.
5. We will not need to overclock the Raspberry Pi, so you can leave this option as it is. Later on in the book we will be interfacing with the Raspberry Pi's display and doing neat tricks with it. For this, we need some memory for the **Graphical Processor Unit** (**GPU**). By default this is set to 64 MB. We will ask for the largest amount of memory possible to be assigned to the GPU, which is 512 MB. Put `512` behind the **GPU Memory** option; there is no need to enter the text `MB`. The memory on the Raspberry Pi is shared between the system and the GPU. Having this option set to 512 MB results in 512 MB available for the system. I can assure you that this is more than sufficient.

6. Now that we are done with the system configuration, we are making sure we can work with the Raspberry Pi. Click on the **Localisation** tab to show the options applicable to the location the Raspberry Pi resides in. We have the following options:
    - **Set Locale**: Where you set your locale settings
    - **Timezone**: The time zone you are currently in
    - **Keyboard**: The layout of the keyboard
    - **WiFi Country**: The country you will be making the Wi-Fi connection in

7. This book uses the US-English language with a broad character set. Unless you prefer to continue with your own personal preferences, change the following by pressing the **Set Locale** button:
    - **Language to en (English)**
    - **Country to US (USA)**
    - **Character Set to UTF-8**

8. Press the **OK** button to continue.

    As this needs to build up the Locale settings this can take up to about 20 seconds to set up. You will be notified with a small window when this process is finished.

9. The next step is to set the timezone. This is needed as we want to have time and dates shown correctly.
10. Click on the **Set Timezone** button and select your appropriate **Area** and **Location** from the drop-down menus. When done, press the **OK** button.
11. To make sure that the text we enter in any input field is correct, we are going to set the layout of our keyboard. There are a lot of layouts available, so you need to check yours. The Raspberry Pi is quite helpful and provides many keyboard options.
12. Press the **Set Keyboard** button to open up a popup showing the keyboard options. Here, you are able to select your country and the keyboard layout available for this country. In my case, I have to select **United States** as the **Country** and the **Variant** as **English (US, with euro on 5)**.
13. After you have made the selection, you can test your keyboard setup in the `input` field below the **Country** and **Variant** selection lists. Press the **OK** button when you are satisfied with your selection.

14. Unless you are connecting your Raspberry Pi to a wired network connection, we are going to set up the country we are going to make the Wi-Fi connection in so that we are able to connect remotely to the Raspberry Pi. Press the **Set WiFi Country** button to have the **Wifi Country Code** shown, which provides us the list of available countries for the Wi-Fi connection. Press the **OK** button after you have made the selection. We are now done with the minimal Raspberry Pi system configuration.
15. Press **OK** in the settings window to have all our settings stored and press **No** in the popup that follows that says a reboot is needed to have the settings applied as, we are not completely done yet.
16. Our final step is to set up the local Wi-Fi chip on the Raspberry Pi.

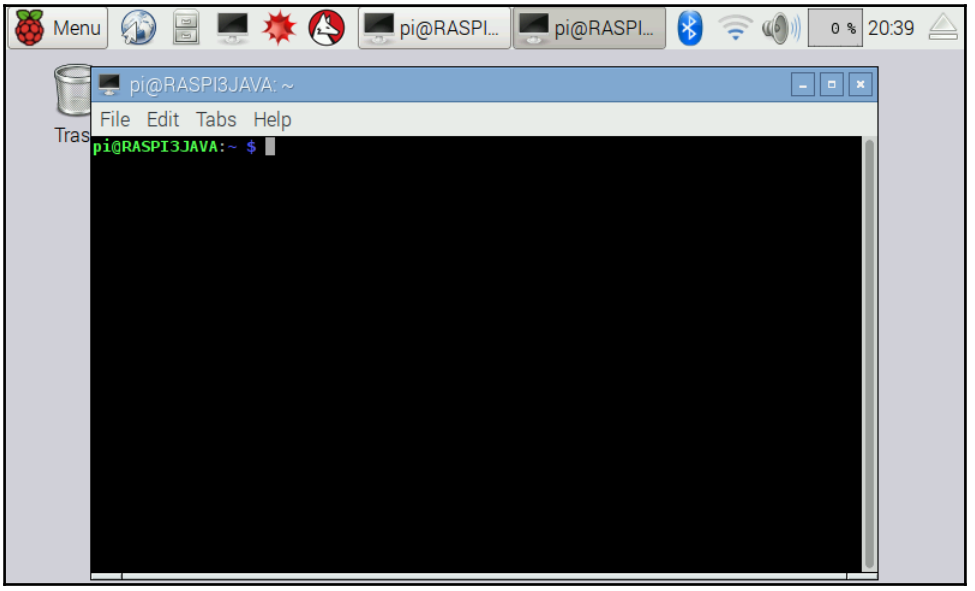
We will now set up the Wi-Fi on the Raspberry Pi. If you want your Raspberry Pi connected with a network cable, you can skip this section and head over to the **Set fixed IP** section:

1. To set up the Wi-Fi, click on the **Network** icon, which is at the top of the screen between the **Bluetooth** and **Speaker Volume** icons.
2. When you click this button, the Raspberry Pi will start to scan for available wireless networks. Give it a couple of seconds if your network does not appear immediately. When you see your network appear, click on it, and if your network is secured you will be asked to supply the credentials required to be able to connect to your wireless network.

If you have any trouble connecting to your wireless network without any messages, log in to your router and change the Wi-Fi channel to a channel lower than channel 11.

3. When you have entered your credentials and pressed the **OK** button you will see the icon changing from the two network computers to the wireless icon, as it it trying to connect to the wireless network.
4. Now that the Raspberry Pi has rebooted, we have configured the wireless network to make sure the wireless network will keep its connection. As the Raspberry Pi is an embedded device targeting low power consumption, the Wi-Fi connection is possible set to sleep mode if after a specific time there is no network usage.
5. To make sure the Wi-Fi does not go into power sleep mode we will be changing a setting through the command line that will make sure that this won't happen. To open a CLI we need to open a terminal. A terminal is a window that will show the command prompt where we are able to provide commands.

6. When you look at the graphical interface you will notice a small **computer screen** icon at the top in the menu bar. When we hover over this icon it shows the text terminal. Press this icon to open up a terminal.
7. A popup will open with a large black screen showing the command prompt, as shown in the following screenshot:



Do you notice the hostname we set earlier? This is the same prompt as we will see when we log in remotely.

8. Now that we have a command line open we need to enter a command to make sure the wireless network will not go to sleep after a period of no network activity. Enter the following in the terminal:

```
sudo iw dev wlan0 set power_save off
```

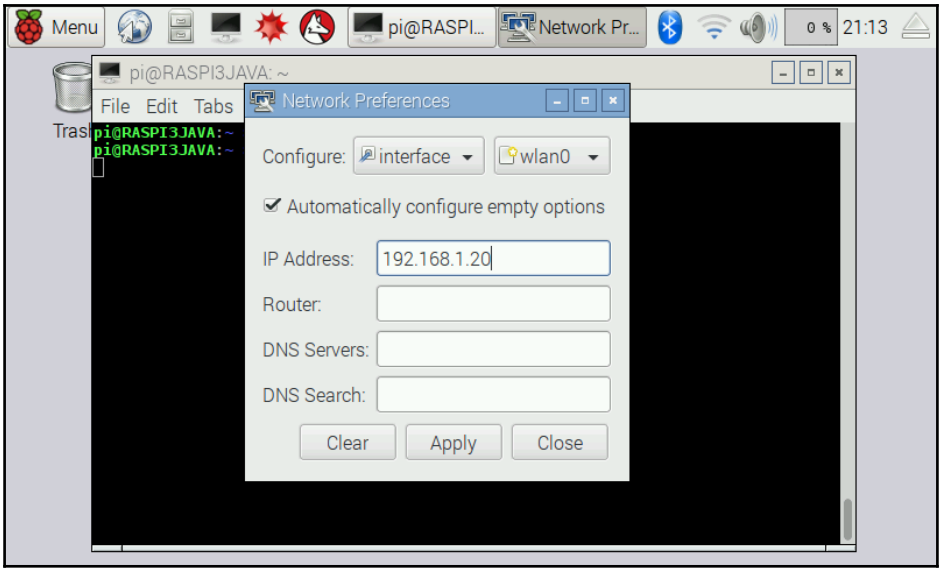
9. Press *Enter*. This command sets the power save mode to off so the `wlan0` (wireless device) won't enter power save mode and will stay connected to the network.

We are almost done with setting up the Raspberry Pi. To be able to connect to the Raspberry Pi from a remote location we need to know the IP address of the Raspberry Pi. This final configuration step involves setting a fixed IP address into the Raspberry Pi settings:

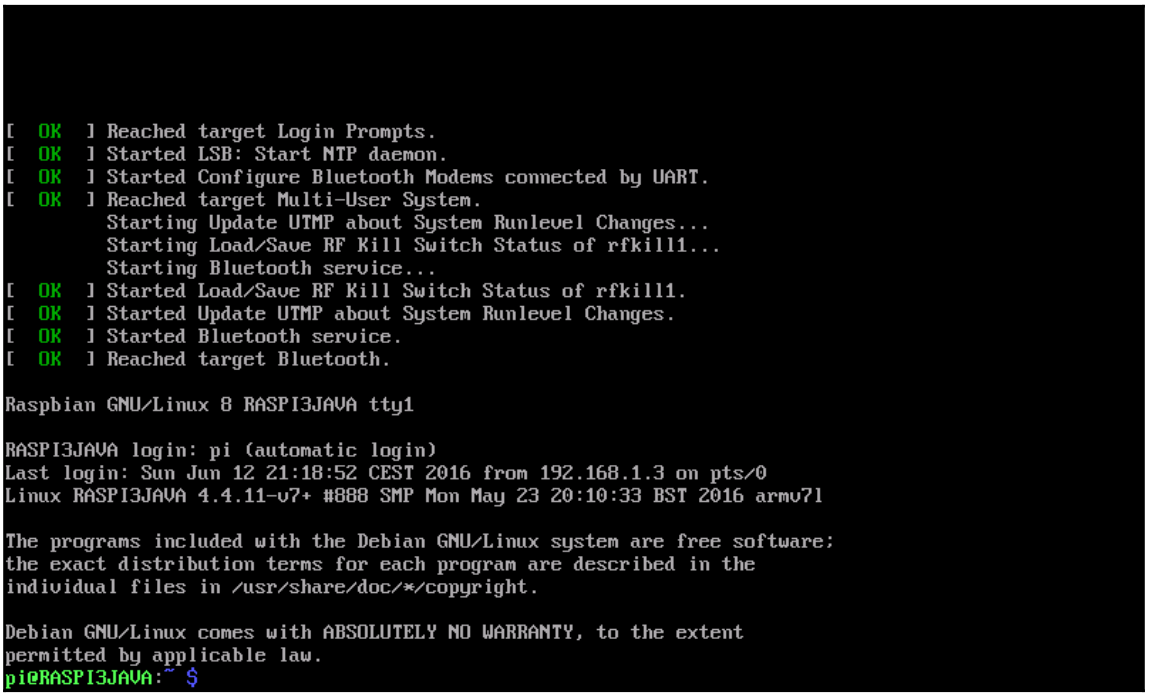
1. To open the settings for a fixed IP configuration, we are going to open up the settings by pressing the wireless network icon with the right mouse button and press the **WiFi Networks (dhcpcdui) Settings** option.
2. A popup will appear with the settings we can change. As we will only change the settings of the Wi-Fi connection, we select I**nterface** next to the **Configure** option. When interface is selected, we select the **wlan0** option in the drop-down menu next to the interface selection.

If you have chosen to use a wired connection instead of the wireless connection you can select the **eth0** option next to the **interface** option.

3. We now have a couple of options available to enter IP address-related information. Please refer to the documentation of your router to find out which IP address is available to you which you can use. My advice is to only enter the IP address in the available fields, which leaves the other options automatically configured, like in the following screenshot. Notice that the entered IP address is the correct one; this only applies to my configuration, which could differ from yours:



4. After you have entered the IP address you can click **Apply** and **Close**.
5. It is now time to restart our Raspberry Pi and have it boot to the **Command Line Interface (CLI)**.
6. While it is rebooting, you will see a lot of text scrolling, which shows the services starting and at the end, instead of starting, the graphical interface, we are now shown the text-based CLI, as shown in the following screenshot:



7. If you want to return to the graphical interface just type in the following command:

```
startx
```

8. Press *Enter* and wait a couple of seconds for the graphical user interface to appear again. We are now ready to install the Oracle JDK, which we will be using to run our Java applications.

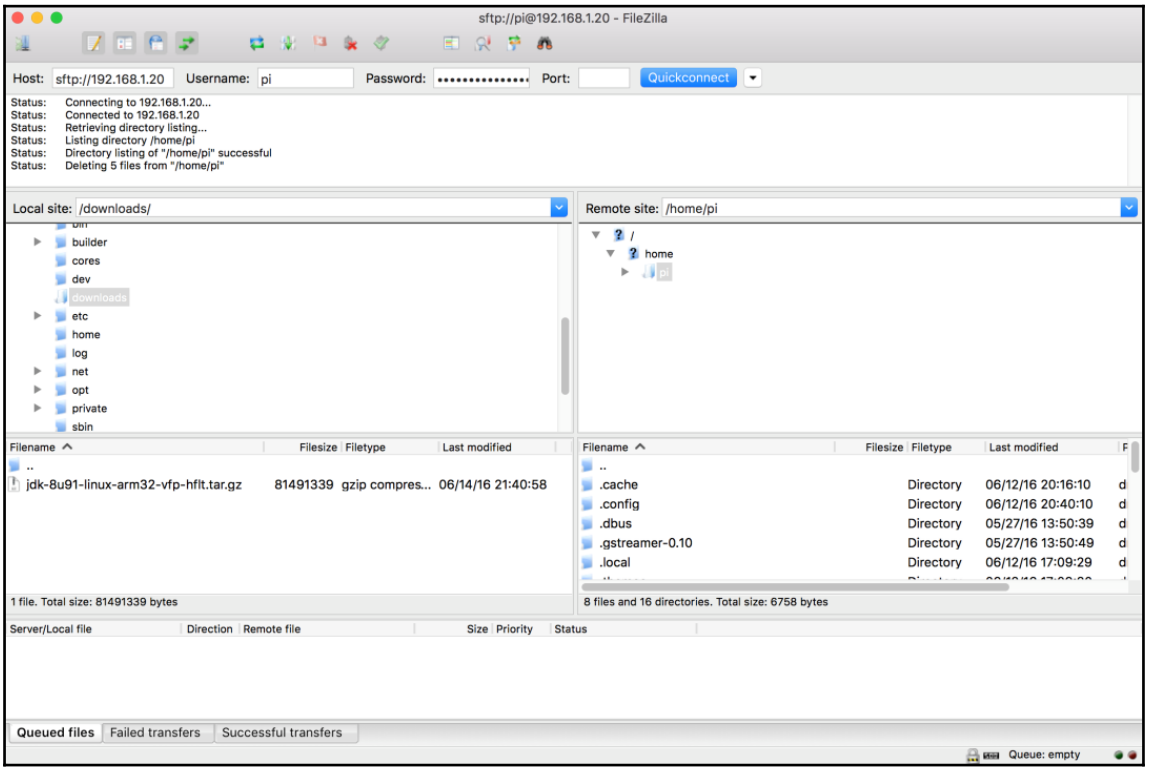
Installing Java

In this section, we will be installing a file transfer application, installing Java, and preparing the NetBeans editor.

We will be using the Oracle JDK to run our Java applications. At the time of writing, the latest version is JDK 1.8, which all our projects will be using as a minimal version:

1. To get the Oracle JDK, go to `http://www.oracle.com/technetwork/java/javase/downloads/index.html` and select the **Download JDK** button.
2. On the following page you need to select the **ARM version** to be downloaded. Although the version you will be seeing now may differ from the version mentioned in the book, the naming scheme for the download is the same in all these versions. Even though the Raspberry Pi is now based on a 64-bit architecture the OS is still 32-bit at the time of writing. This also means the version you will need to download is: `jdk-8u[buildnumber]-linux-arm32-vfp-hflt.tar.gz`, which is the 32-bit version. The `[buildnumber]` indicates the release build of the JDK. All projects are targeted to work on JDK8, so any build number that is at least `91` or higher should be working. At the time of writing, the JDK version used is `jdk-8u91-linux-arm32-vfp-hflt.tar.gz`.
3. To be able to use this Java version we will be using an application to upload the downloaded version to the Raspberry Pi. The application we will be using is called **FileZilla** and it is available for Windows, Linux, and Mac. To download this application go to `https://filezilla-project.org/` and by clicking the **Download FileZilla Client** link, the application will be downloaded.

4. To install FileZilla open the downloaded installer and follow the instructions on the screen. We will now upload this Java version to the Raspberry Pi. Open FileZilla, and you will be presented with the window shown in the following screenshot:



5. To connect we need to enter the credentials that we have set during the installation of the Raspberry Pi. In the most upper part below the icons we will enter the credentials, which are as follows:

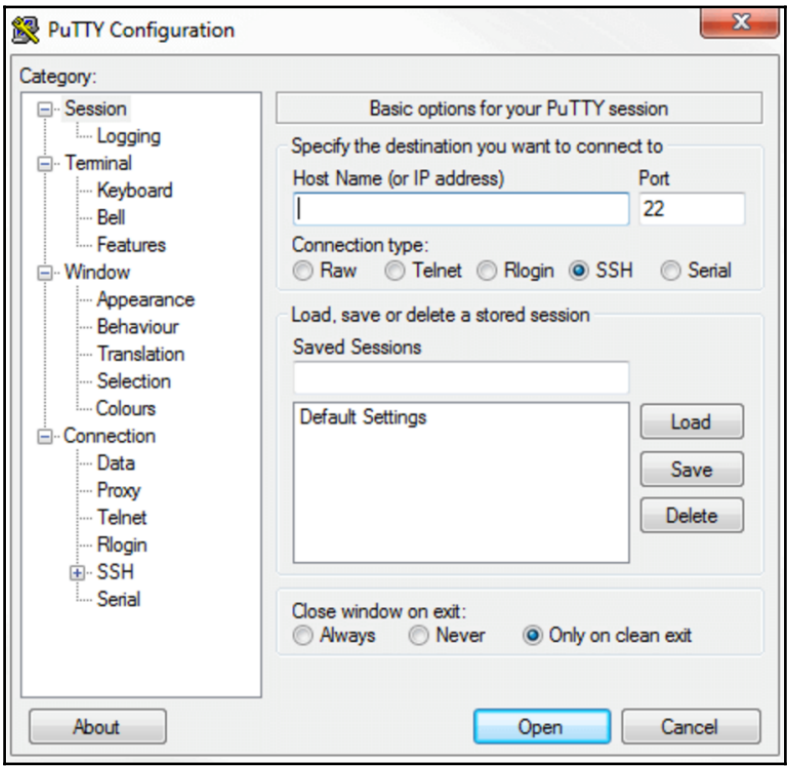
- Only the IP address in the `Host` field
- The username `pi` in the `Username` field
- The password we have changed during preparation in the `Password` field
- We enter `port number 22` in the `Port` field, which is the secured connection

6. Press the **Quickconnect** button, and when all the credentials are entered correctly we are logged in to the Raspberry Pi. On the left side in FileZilla we see the structure of our local computer, and on the right side we see the directory structure of the Raspberry Pi.
7. Browse to the downloaded Java version on the left side of the application. When you see the Java version that we downloaded click and move this file from the left to the right side. The file will now be uploaded to the Raspberry Pi.
8. In the most bottom part of FileZilla we see the upload progressing. When the upload is finished, the progress is removed from this list and the upload is done.

   We can now close FileZilla and do the final steps required to install this Java version.

9. For this, we will use the CLI on the Raspberry Pi. There are different ways to get to this command line. On Windows, we can use PuTTY. This is a small application that allows us to log in from the Windows PC onto the Raspberry Pi. On a Mac we can use the terminal application to log in. When we have our keyboard and display connected to the Raspberry Pi we are already logged in. As we will be using the Raspberry Pi as a remote device, we will be using PuTTY on Windows and the terminal on Mac. When you use Mac, you can skip the paragraph where we will be using the terminal.
10. On Windows, we use PuTTY to connect to the Raspberry Pi. To use PuTTY we need to download it from `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.
11. On the second half of this page, we see a list of files ready to be downloaded below the **Binaries** header.

12. Click the `putty.exe` file to download. This file does not need to be installed, so we can save this file in any location that we can easily reach. When PuTTY is downloaded, double-click it to open it so that you see something similar to the following screenshot:



13. With this window open we can enter the IP address of the Raspberry Pi in the **Host Name** field.
14. We would like to save this IP address with a recognizable name, so we enter **RASPI3JAVA** without the quotes in the **Saved Sessions** field and press **Save**.
15. We now have saved this session for future usage. When we now press **Open** we are prompted with **Login as,** where we enter the username **pi** and press *Enter*.
16. The next step is to enter the password we have set during the preparation. When the credentials are entered correctly we are logged in.

17. To log in on the Mac we use the terminal application, which is preinstalled on the Mac. We open this application by pressing **command** + **space** bar at the same time. This will open **Spotlight Search,** where we can type `terminal` without quotes and press *return* or *Enter* on the keyboard to open the terminal. Hold the **option** button while clicking on the **green circle** in the top-left corner to maximize the terminal window.
18. We are now in the CLI on the Mac. To log in on the Raspberry Pi we need to enter the following command:

```
ssh pi@192.168.1.20
```

The connection we make is done with `ssh`, which is a small terminal application that creates a secure connection with the Raspberry Pi. We use `pi` as the username where the `@` character literally means `at` because we log in as this user at the specified IP address. In the preceding command-line example, I have used the `192.168.1.20` IP address of my Raspberry Pi. You should replace this with the IP address you have set for yours. We are now asked to enter the password; enter the password that has been set for the user pi during the preparation.

From this point on it does not matter how we are logged in on the Raspberry Pi as the following steps are the same via Windows with PuTTY, Mac with terminal, or using the keyboard with a display attached to the Raspberry Pi. We are now presented with the CLI on the Raspberry Pi, which gives us the opportunity to unpack and install the Java package we have uploaded to the Raspberry Pi. We will place this installation in the `/opt/` directory. To make this possible we first create the `java` directory inside the `/opt/` directory by entering the following command:

```
sudo mkdir /opt/java
```

We use the `sudo` command because this directory is owned by the `root` user, which is the administrator on the Raspberry Pi. The `sudo` command temporarily leverages our privileges to be an administrator for the duration of the command. With `mkdir /opt/java` we created the `java` directory inside the `/opt/` directory. Now that we have the `/opt/java` directory we are going to unpack the downloaded Java archive file. We do this by entering the following command:

```
sudo tar -C /opt/java -xf jdk-8u91-linux-
arm32-vfp-
hflt.tar.gz
```

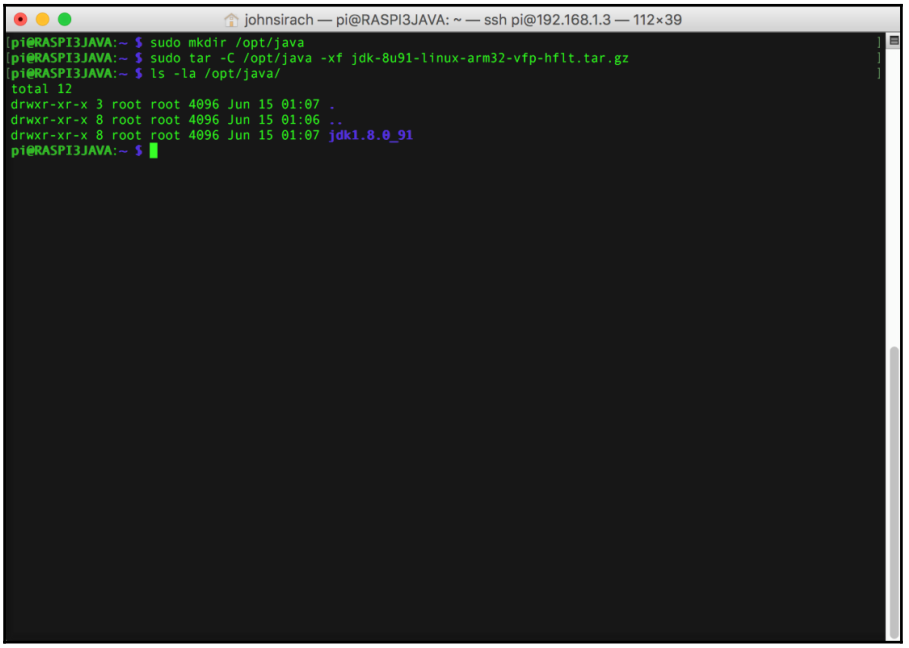
Again we have used the `sudo` command to leverage our permissions. With the `tar` utility we unpack the Java archive into the `/opt/java` directory with the help of the `-C` parameter line switch. The command line parameters `-xf` tells the tar utility that we want to unpack the `jdk-8u91-linux-arm32-vfp-hflt.tar.gz` file. The filename in the preceding example is the version downloaded at the time of writing. Replace `91` with the version you have downloaded.

The Linux command line has a nice way to help you enter commands, directories, and filenames. When you have entered a part of the filename pressing the *Tab* key will auto-complete the filename.

We can now check if the files have been extracted correctly from the downloaded Java archive. By entering the following command we should see the directory extracted from the archive:

```
ls -la /opt/java/
```

If you see the `jdk1.8.0_91` directory, as shown the following screenshot, we have successfully unpacked Java. If you have a different version of the JDK, you will see a different number than `91` at the end:



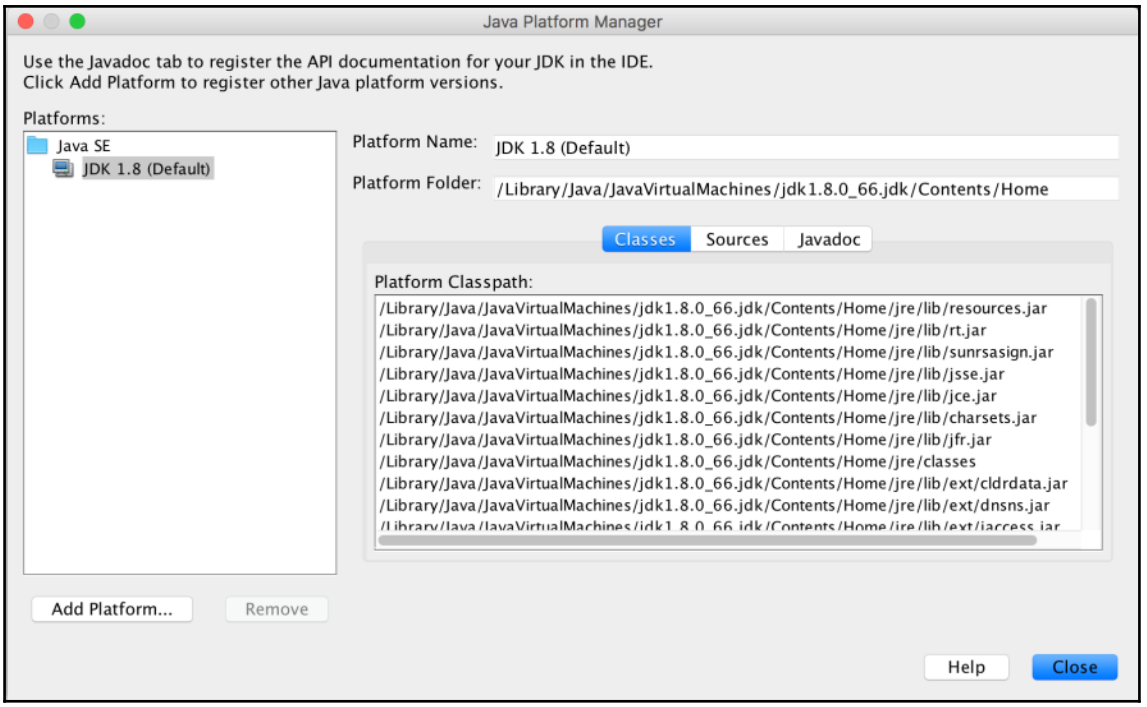
Installing and preparing the NetBeans Java editor

At this point we have installed and prepared the Raspbian Linux OS and downloaded and installed Java JDK 1.8 on the Raspberry Pi. Our final step is to install and prepare NetBeans with Java as our editor, which is also used to upload and run the applications on our Raspberry Pi. We will be using NetBeans 8.1 as a reference for installing and preparing.

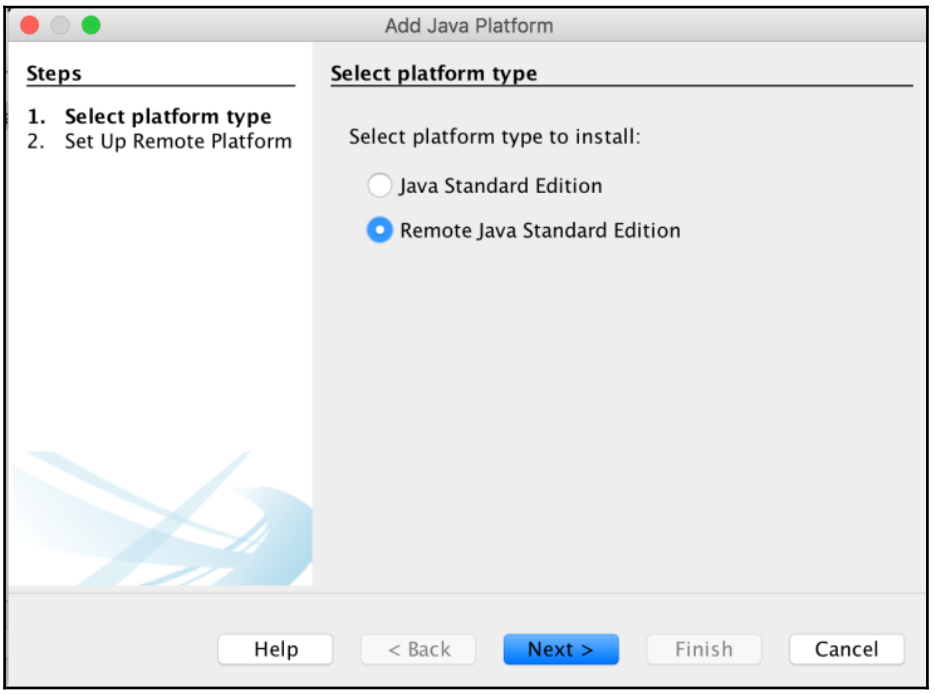
If you are not used to using NetBeans; no problem, we will go step-by-step with downloading and installing and preparing the application. To install NetBeans on your computer we first need to download the correct version onto our PC or Mac. As NetBeans runs on Java, it is also available for Linux-based OSes. To get NetBeans, go to `http://www.netbeans.org` and press the **Download** link on the top right of the page. You will be redirected to the download page, where you will see three download options. As we are focused on Java SE, the first and smallest download will suffice for our projects. Press the **download** button for the left-most download to download the smallest version. After download is completed, we need to install it. The installation procedure is almost identical for all platforms. All platforms have their dedicated installer, and the installation instructions are linked to on the download page below the show download bundles.

When you open NetBeans for the very first time you will be presented with a welcome screen. You can set a mark in the top-right corner of this welcome screen that says **Do not show this again**. Otherwise it will show up every time we start up the editor. We will now configure NetBeans in such a way that we are able to write our application on our local computer, and when we want to run our code, let NetBeans upload the application to the Raspberry Pi and run it. We will be able to see all the output we normally would see when we log in on the Raspberry Pi and run the Java application in the output window of the NetBeans editor! This will speed up our projects a lot as we do not need to do any manual actions such as uploading and logging in. To make use of this very handy feature we need to configure a **Remote platform**. To configure the remote platform with the Raspberry Pi, we will adding it to NetBeans.

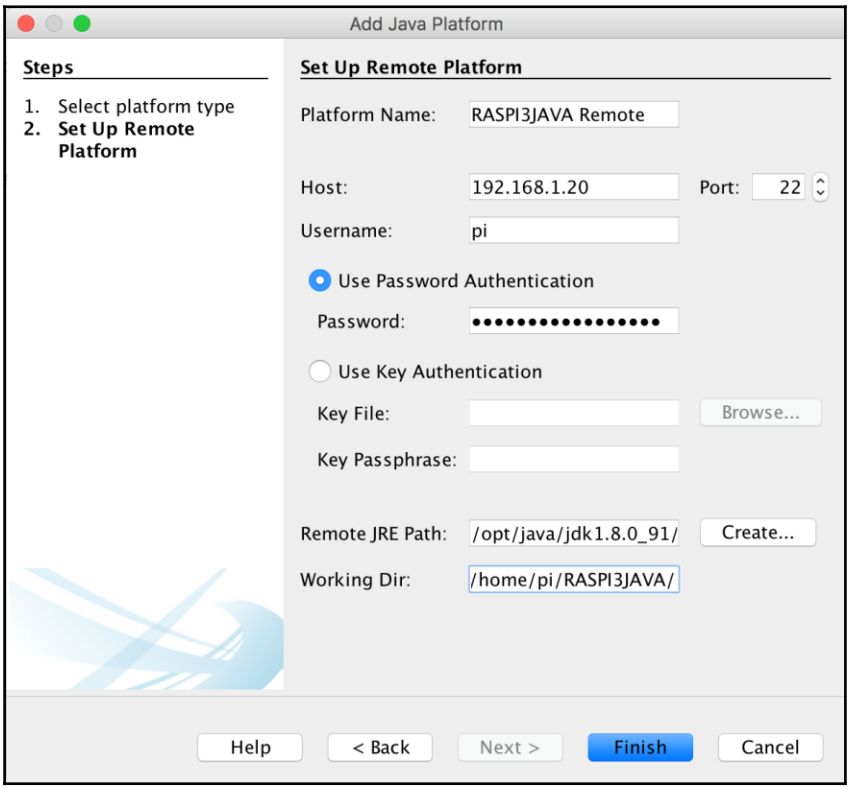
First, we will be defining the platform itself. In NetBeans, click **Tools** | **Java Platforms**. The **Java** platforms settings dialog will show up:



On the left bottom, click on the **Add Platform** button. We are now presented with a selection if we want to add a local or a **Remote Java Standard Edition**. We select the **Remote Java Standard Edition** and press **Next**:

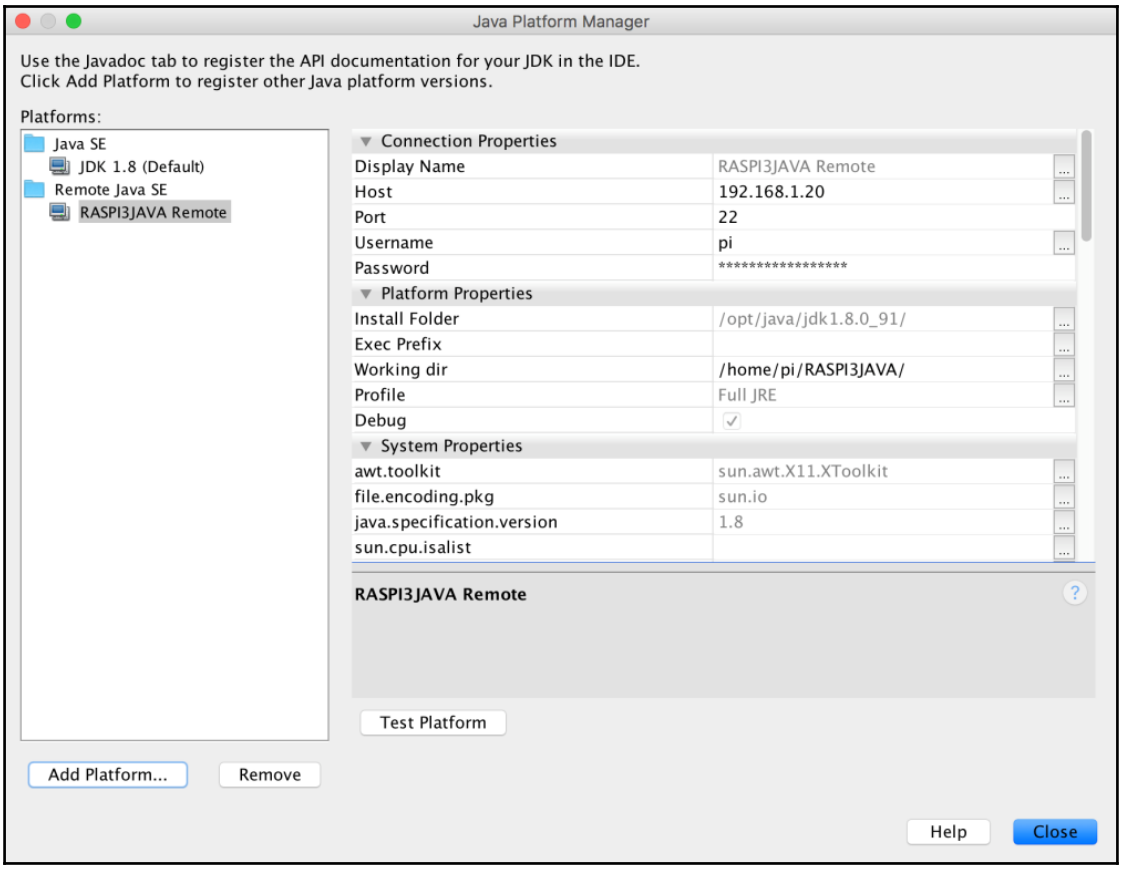


We are now presented with a follow up popup were we can enter the details of the remote platform, which is installed on the Raspberry Pi. In the following screenshot, examples are filled in, which we can use for the remote platform:



We just replace the **Host** IP address with the address that the Raspberry Pi is reachable on. We keep the **Username** pi and enter the **Password** we set during the preparation. We set the path to the JDK installation according to the version we have installed. In the example, it is JDK1.8 build **91**. Change the **91** to the build you have installed. We do not press the **Create** button as the directory is already there. As the working directory, we keep the `/home/pi/RASPI3JAVA/`, so that we know where all our projects will be placed.

Press the **Finish** button to have NetBeans verify the remote platform and have it download properties from the remotely installed Java version. This action will take a couple of seconds. When you see the following screen, the remote platform has been confirmed by NetBeans:

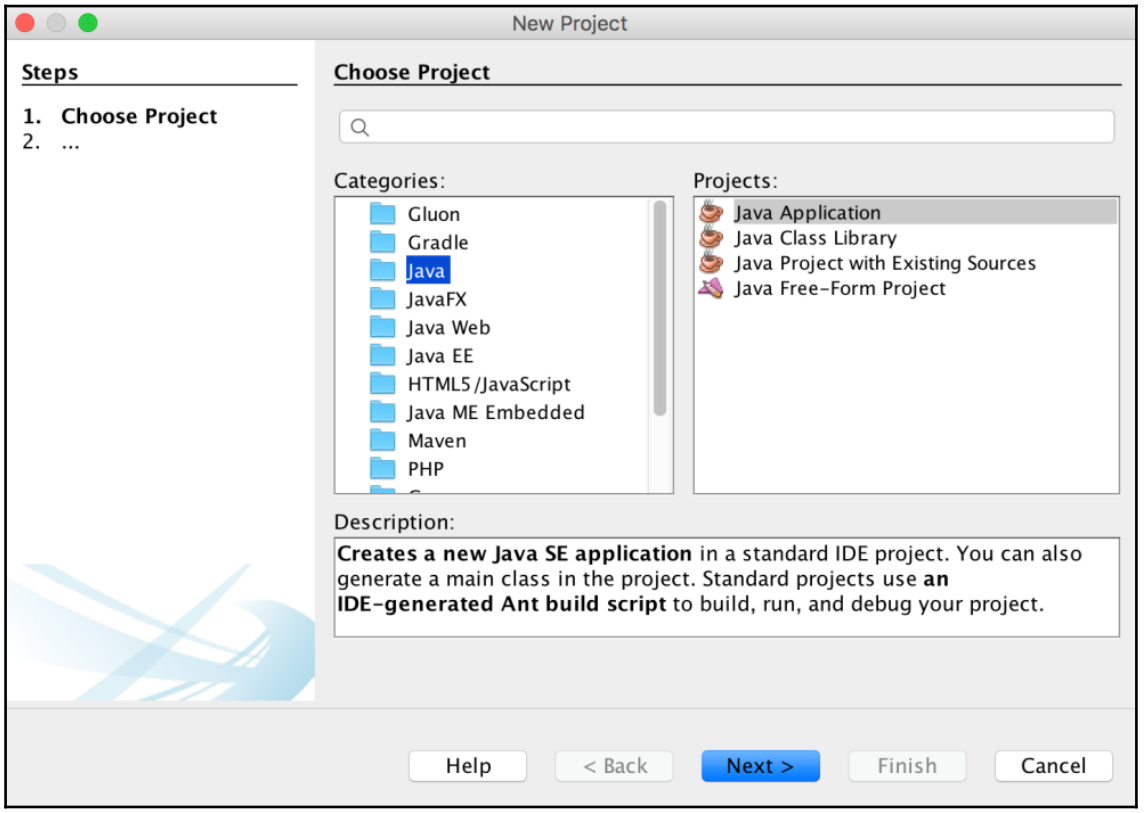


We press **Close** to close the dialog so we can begin with our projects.

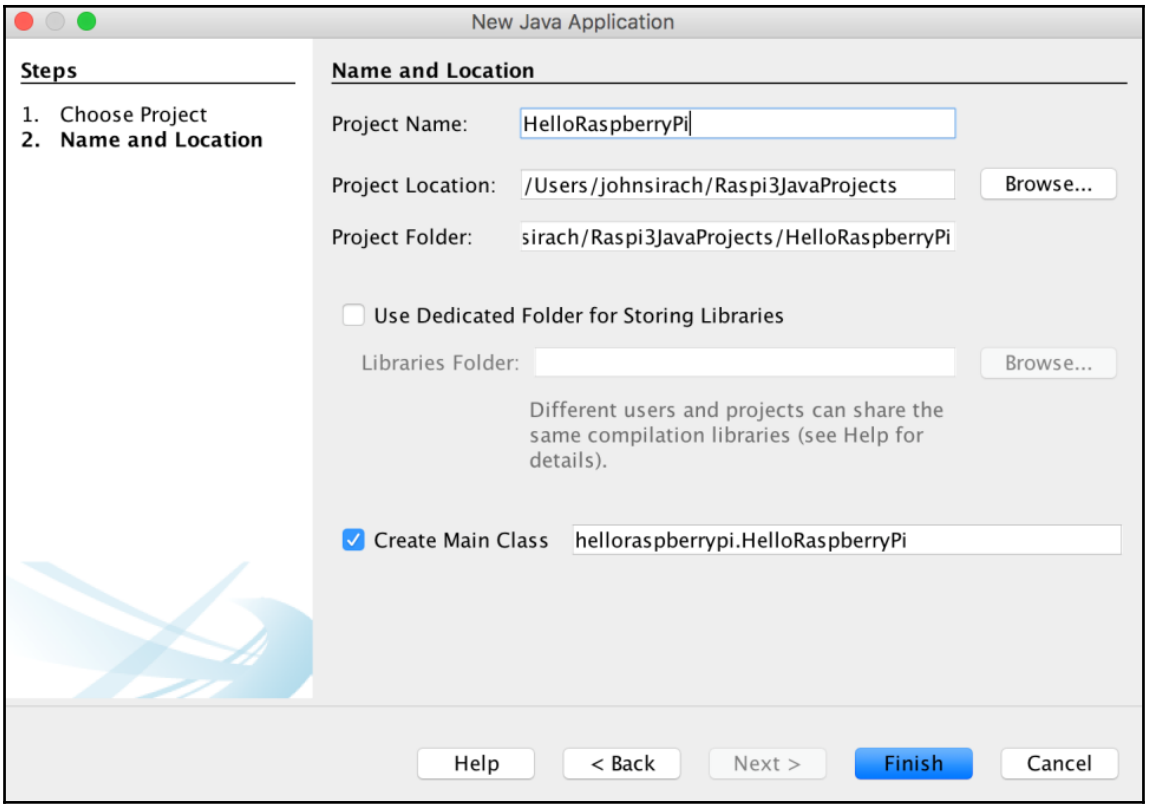
Our first remote Java application

Now that we have added a remote platform, we will be writing a very simple little Java application to check that it works and have the output from the application executed on the Raspberry Pi and output its information in the editor. We start with a simple Hello World Java application and run it on the Raspberry Pi. The following steps will be needed every time we start a new project.

Click **File** in the menu bar and click **New Project**. You will be presented with the New Project wizard, as shown in the following screenshot:

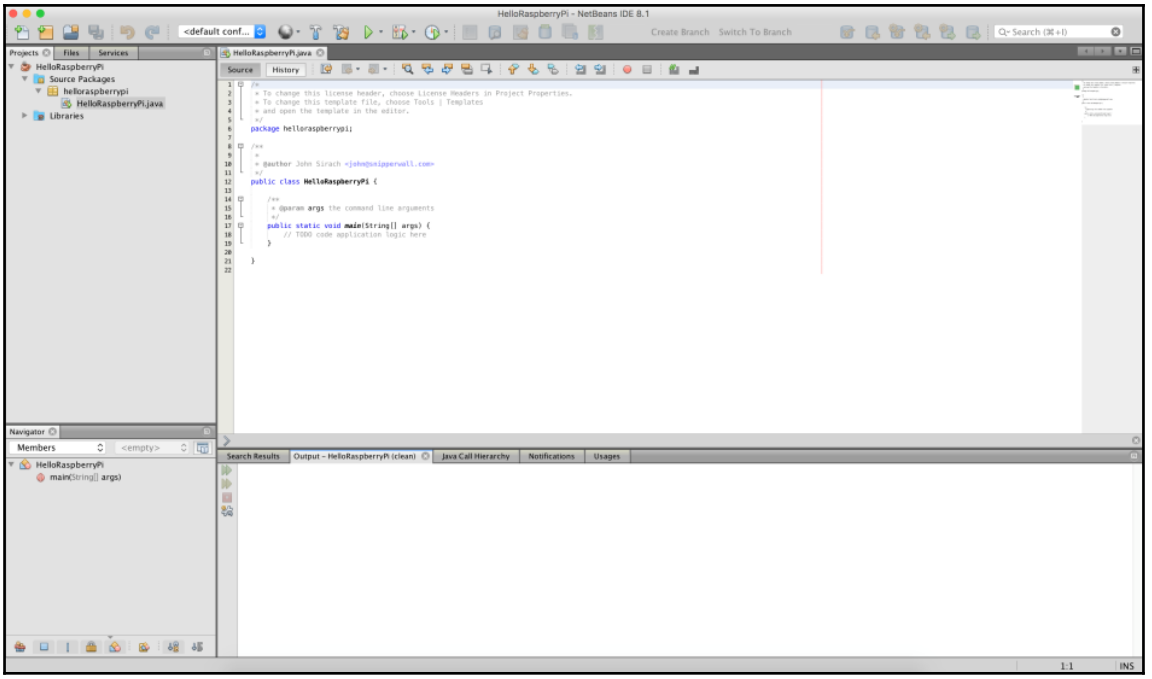


On the left side are the project categories. Here we click on `Java`, and on the right side we select the **Java Application** option and press **Next** to create a Java application. To be able to start with a Java application we need to enter some details so we can begin with our development. The following screenshot shows the details we need to enter to begin:

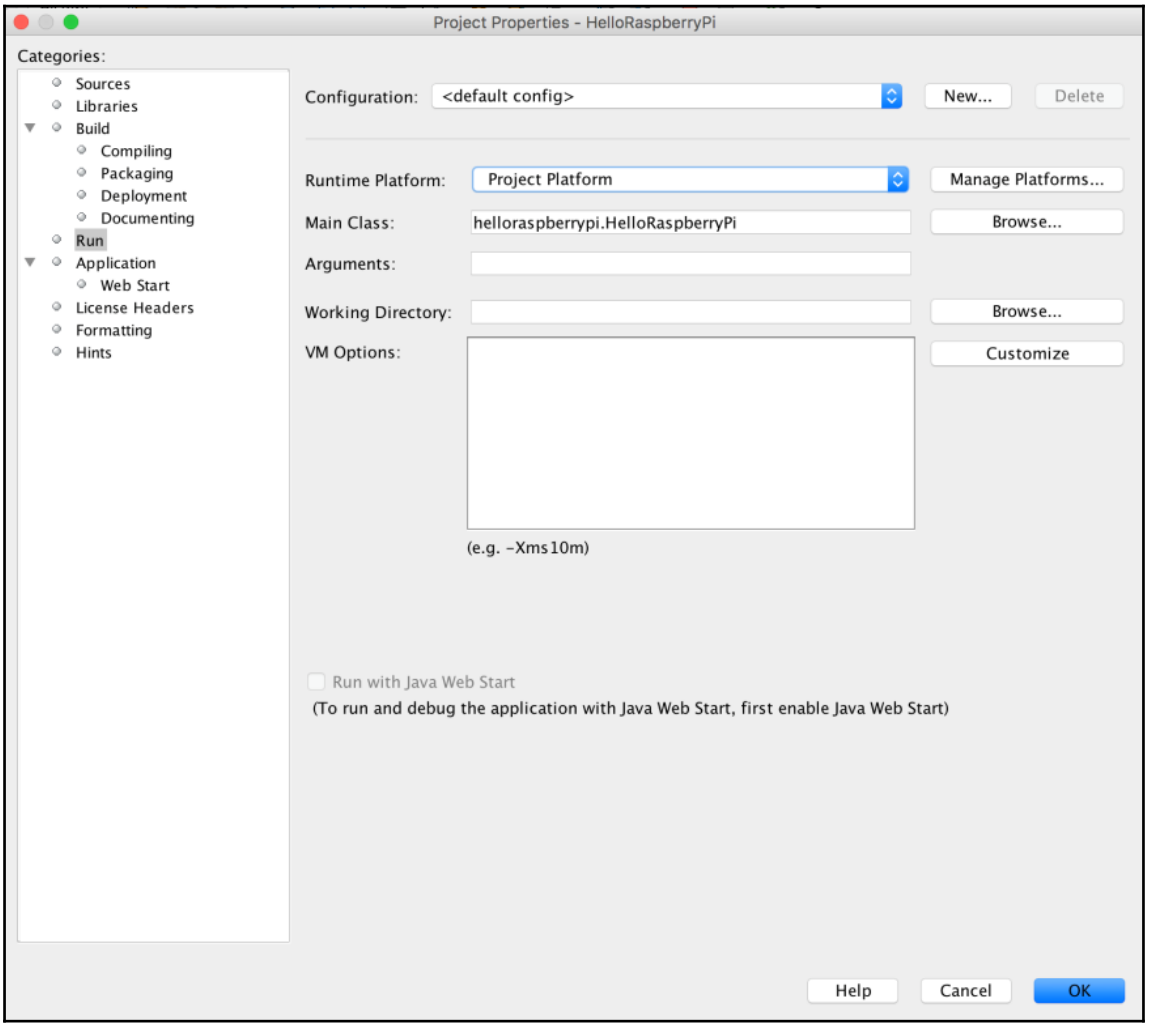


In this screen we give the project the name, `HelloRaspberryPi`, and browse to the `local` directory where we will be putting our project. Make sure you put no spaces in the name of the project! We supply a base path where this project is being placed. In my case, I will place my project in my `home` directory with the `sub`director `Raspi3JavaProjects`. Other projects that we will be creating later on will also be placed in this directory. NetBeans will automatically create a directory based on the project name in this base directory. Click **Finish** and the project will be created.

On the left side we have our project structure tree where all our Java packages are, and on the left side we have the editor. This window will be our main projects screen throughout the book, as shown in the following screenshot. This window is the same on all platforms, such as Windows, Linux, and Mac:



Now that we have a new project, we will configure it to run on the remote Raspberry Pi platform. To do this we open the project properties by clicking on the project name in the left side of the window. This opens up a context menu where the bottom option is **Properties.** When clicked, it opens the Properties window, as shown in the following screenshot:



We click on the **Run** configuration in the left option tree. We now see the possible run options. We want to run our Java application on the Raspberry Pi, so we change the `Runtime Platform` to `RASPI3JAVA` remote or the name you have set in the remote platform configuration. We are asked to save this as a configuration, and we enter a name such as remote Raspberry Pi configuration. Press **OK** to save our run configuration.

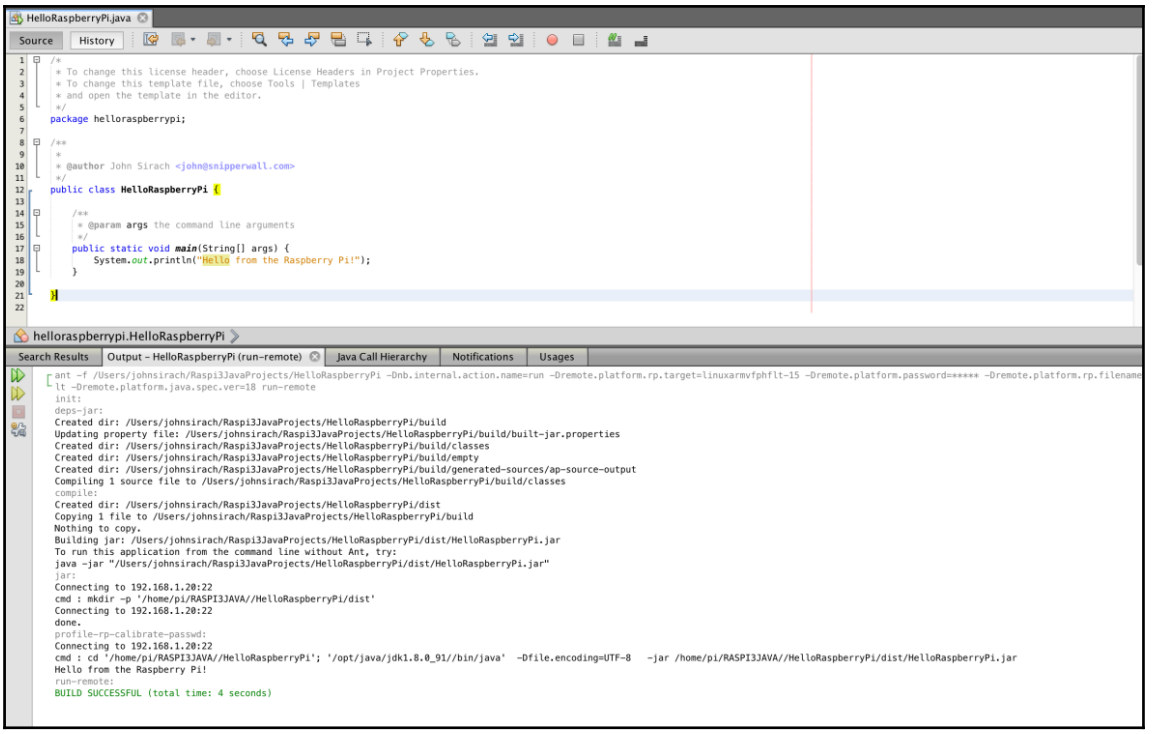
Running our application on the Raspberry Pi

Now we are ready to write a line of code and have it run on a fully prepared Raspberry Pi with Java and a fully prepared NetBeans Java editor. We will change the main run function of our Java application to have it print out `Hello from the Raspberry Pi!`. Change the code in the main function so that we have what is shown in the following snippet:

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    System.out.println("Hello from the Raspberry
Pi!");
}
```

For now, this is enough. This will print out the line stated previously. To run this on the Raspberry Pi we have three options. One is to click on the **Run** button in the button bar below the M**enu** bar; the second is to right-click on the project name and select **Run** from the context menu; and the third is to press *F6*. Pick the one easiest for you.

Now the application is compiling and will be executed on the Raspberry Pi. The output should look same as following screenshot:



This is one of the many reasons why NetBeans is my personal favorite editor. When we look at the output we can see NetBeans building the application and logging in to the remote platform, which in our case is the Raspberry Pi. When logged in it creates a directory called `/home/pi/RASPI3JAVA/HelloRaspberryPi/dist`. This is the directory the `jar` file is placed, so it can be executed on the Raspberry Pi. We can see this application being executed by the CLI command: `cd '/home/pi/RASPI3JAVA//HelloRaspberryPi'; '/opt/java/jdk1.8.0_91//bin/java' -Dfile.encoding=UTF-8 -jar /home/pi/RASPI3JAVA//HelloRaspberryPi/dist/HelloRaspberryPi.jar`

What this does is `cd '/home/pi/RASPI3JAVA//HelloRaspberryPi',` which makes sure the directory is the correct one. This is followed by the `'/opt/java/jdk1.8.0_91//bin/java' -Dfile.encoding=UTF-8 -jar /home/pi/RASPI3JAVA//HelloRaspberryPi/dist/HelloRaspberryPi.jar` command, which runs the Java executable with our `jar` file as input. All the directories we have created and the installed java version is used. Our result is shown with: `Hello from the Raspberry Pi!`. We can now start with our projects, well done!

Summary

In this chapter you learned essential techniques on how to prepare the Raspberry Pi 3 ready to use with projects will be discussing through out this book. Then you installed Raspbian on the Raspberry Pi 3 followed by Java, installed NetBeans IDE on your computer and connected the Raspberry Pi with the local computer through SSH using PuTTY. Finally, you wrote a simple Java application, deployed it to the Raspberry Pi and run.

In `Chapter 2`, *Automatic Light Switch Using Presence Detection*, you will use this basic Raspberry Pi setup to build a automatic light switch which uses presence detection. So you will learn how to expand this basic Raspberry Pi setup by adding some hardware and software components.

2
Automatic Light Switch Using Presence Detection

Now that we have our Raspberry Pi 3 up and running we can start with our first project that uses it. In this chapter, we will be doing presence detection using the integrated Bluetooth chip and show it on a HD44780 16x2 display. As only showing presence will not fill the whole display, we will be adding light intensity detection and environmental temperature. We would also like to add a home automation factor to this setup by adding a relay that can turn on a light. This little project could possibly be placed at a strategic location such as an entry hallway on which we will focus.

We would like to turn on a light switch when we're nearly home when the light in the hallway is quite dim and show the current temperature. This chapter will then cover the basics of using libraries in other projects in the book.

To be able to build this, we will be covering the following topics:

- Introduction to and installing Fritzing
- Billing of materials
- How to emulate reading analog values on digital pins
- Start our project and install the necessary libraries
- The `Pi4J` libraries
- Adding the HD44780-compatible 16x2 character display
- Showing data on the HD44780-compatible display

- Adding the **Light-Dependent Resistor** (**LDR**) to the setup
- Reading and displaying values from the LDR
- Using digital out to switch to switch and display the relay status
- Automatic switching based on environment lighting
- Using the Bluetooth chip on the Raspberry Pi
- Bluetooth device discovery
- Putting it all together, our first automation project

Introduction to and installing Fritzing

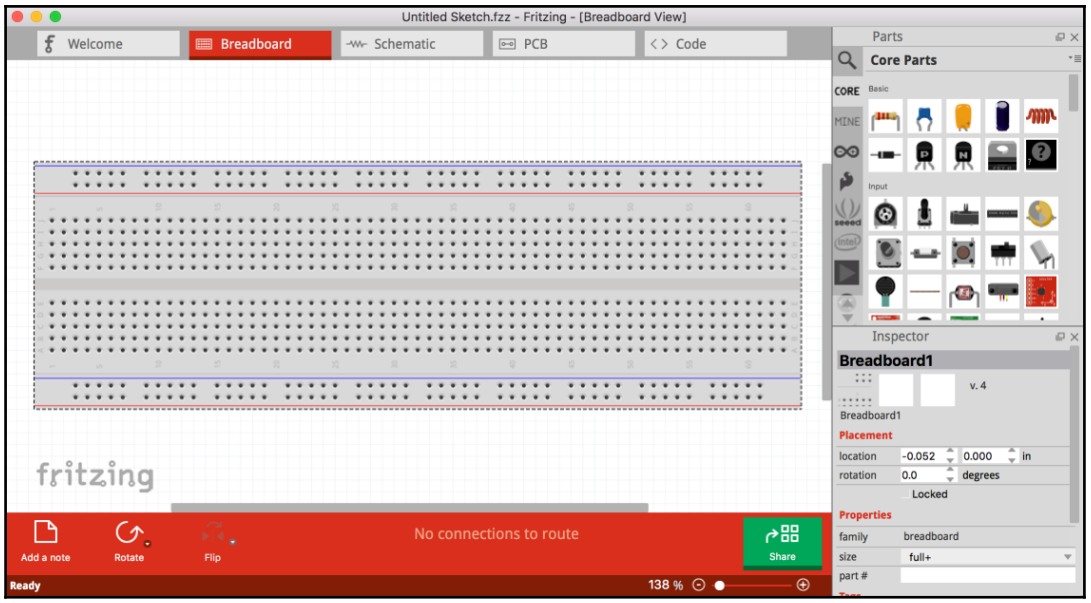
If you already know about Fritzing, you can skip this section and continue to the Bill of Materials section.

Fritzing is one of the most popular and most commonly used free and open source electronic schematics designers used by electronics hobbyists. This application also makes it very easy for beginners, as schematics are built using drawings from actual devices.

The community surrounding this application is very large, resulting in a lot of schematics made available for download and electronic parts being added at a large rate. I advise that after reading the book, you take a closer look at the community part as you will be able to build a lot of applications with the knowledge gained.

To install Fritzing, all we need to do is go to the Fritzing website at `http://fritzing.org` and click the **Download** link on the top of the website. The installation is explained in full detail on the Fritzing download page for Windows, Mac, and Linux.

When Fritzing is installed and opened, we are presented with a welcome window showing some different options. At the top of the window there are some buttons that, when Fritzing is started has the **Welcome** button selected. On the same level, there is a **Breadboard** button. When clicked, we see a breadboard that we will be using to build up our schematics, as shown in the following screenshot:



In the previous screenshot, we see the breadboard on the left. This will be our workplace, where we are going to draw connections between all the components. On the right side, all the available parts are displayed with their respective component options. With every component placed on the breadboard we can set the options. For example, when we place a resistor on the breadboard, we are able to change the resistance, which will be reflected on the component.

Next to the screenshots, the file with the Fritzing schematics will be available for download, which is included in the zip file and also has the code available. As Fritzing is a community-driven application, it is possible some components are not available for use in our projects. Don't worry, as the download also includes the Fritzing parts that could be missing from your Fritzing download. When you unpack the download, you will notice a folder called `B05688_02_schema`. This does not only contain the schematic we use, but also all the possible missing components. To install these, open the Fritzing application and do the following:

1. Click on the drop-down menu next to the **Core Parts** header we see when opening the application in the upper-right corner.
2. A context menu will appear, and when clicking on the **Import** option, a new popup will appear.

3. In this popup, browse to the folder containing the components and select them all.
4. Click on the **Open** button.
5. All components selected will now be available for us to use.

Billing of materials

To be able to realize our build, we need the following components:

- A breadboard
- A Raspberry Pi, of course
- A Raspberry Pi 3 T-Cobbler or male to female jumper wires
- Breadboard wires
- Three 10 Kohm resistors
- A Hitachi HD44780 16x2 character display
- One 10 Kohm potentiometer
- One 4.7 Kohm resistor
- A 1 μF capacitor
- A LDR a.k.a. LDR
- One BC546B NPN transistor
- One 5V switchable mechanical relay, such as a Keyes 5V relay module or similar
- One 1N4148 diode
- A device capable of being detected by Bluetooth, such as a mobile phone

How to emulate reading analog values on digital pins

Let us start with some theory. The Raspberry Pi only has digital pins. This results in being unable to read analog values. But with some devices, we are able to read analog values, or rather, create an **RC Circuit** to cheat a little bit. An RC circuit is a circuitry setup where we place a resistor and a capacitor in series with each other. When a voltage is applied on the circuit, the voltage across the capacitor will rise. The higher the resistor value is, the longer it will take for the voltage to rise across the capacitor.

With a fixed resistor, the time to equalize the voltage before and after the resistor around the capacitor will be the same. When we introduce an LDR, where the resistor value depends on the light intensity, the more light that hits the sensor, the lower the resistance is. With this in mind, we take a look at a feature on the digital pins on the Raspberry Pi.

The digital pins on the Raspberry Pi are capable of detecting a low and a high voltage value. A low is when the voltage is lower than a threshold, and a high is when the voltage is higher than the threshold. This threshold is called the edge, and on the Raspberry Pi, the edge is around 1.3V. Now that we know that the Raspberry Pi is able to detect these values, we can use this.

We will be creating a circuit where we utilize this feature and characteristics. In our circuit, we will connect the components such that we are able to completely drain the capacitor and charge it back up where at a certain point the rising edge on the Raspberry Pi is detected. This setup, combined with an LDR where the capacitor will be charged depending on the light intensity, causes different RISE times. The longer it takes a RISE time to be reached, the darker it is. Let's start setting up our project.

Starting our project and installing the necessary libraries

This project is available for download and can be opened within NetBeans. The only thing we need to do is change the project's properties so it is able to be run on the Raspberry Pi. When the project is opened, go to the project's **Properties** and by selecting the **Run** node on the left side, we can change the **Runtime Platform** to the Raspberry Pi platform. This is explained in detail in `Chapter 1`, *Setting up Your Raspberry Pi*, in the *Our First Remote Java Application* section.

For our project, we need a couple of libraries. We will be using the libraries from the `Pi4J` project that enable us to interact with the Raspberry Pi pins and use BlueCove libraries to be able to interact with the Bluetooth chip.

Although the libraries are available in the project, it will be useful to know how these are added to the project if you start afresh. This will be valuable when you want to extend any project with your own code.

When the project is opened, there is a node called **Libraries** that is in the project structure tree on the left side. Right-clicking on this node will show a context menu where you can select **Add Project**, **Add Library**, and **Add JAR/Folder**. The **Add Project** option gives you the possibility of adding another project to your project as a resource, **Add Library** will give you the option from a set of available libraries, and the **Add JAR/Folder** option gives you the option to select a folder containing JAR files or a single JAR file. My advice would be to put all the needed libraries in a single folder and add the folder to the **Libraries**.

The Pi4J libraries

The `Pi4J` library is developed by Robert Savage and is used to interact with the GPIO pins. Communication with the GPIO pins is not done by pure Java but uses a native library developed by Gordon Henderson. This library is bundled within the `Pi4J` library and is used automatically. The `Pi4J` project consists of multiple libraries which are as follows:

- `pi4j-core.jar`: Always needed, and includes the `pi4j` native library
- `pi4j-device`: Wrappers used for interactions with different kind of peripheral and different Raspberry Pi expansion boards
- `pi4j-gpio-extension.jar`: Used by `pi4j-device.jar` or for custom extension implementations
- `Pi4j-service.jar`: Provides listeners for GPIO pins

We will always include all libraries so we are sure to have all the dependencies and will make it easier for you to extend the projects without the need to search for the corresponding libraries of the same version being used within our projects. All the projects we will be going through will use the `Pi4J` libraries.

More information about the `pi4j` libraries can be found at `pi4j.com`.
More information about the wrapped native library called `wiringpi` can be found at `wiringpi.com`.

Adding the HD44780-compatible 16x2 character display

Let's start our project by connecting the 16x2 character display to the Raspberry Pi and show some text on it using Java. When looking around on the Internet for a 16x2 character display, this will often be a HD44780-compatible one. These displays are quite affordable and are used in a lot of projects. Before we attach this display to the Raspberry Pi, we need to check what kind of backlight is being used as there are both LED and EL types. In our schematic and connections, we will be connecting a LCD with an LED backlight.

The character display has a parallel interface, which means we will need a lot of pins to connect the display. We will be communicating in 4-bit mode with the display so we do not need to use all the pins that are available. The pin setup of this kind of display is mostly as follows, ordered by pin number:

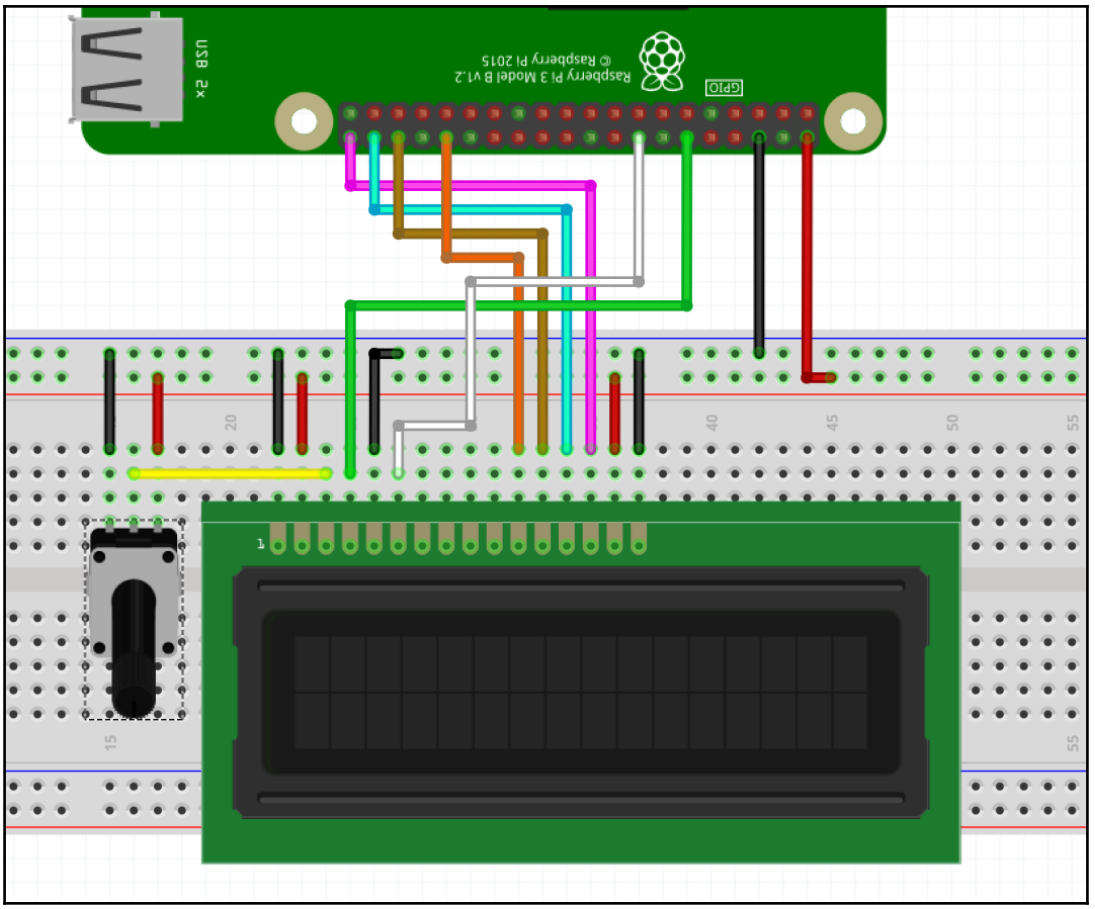
- Ground
- 5V VCC, (not 3.3V)
- Contrast (VO) from a potentiometer
- Register Select (RS) (LOW=command, HIGH=data)
- Read/Write (R/W). (LOW=write, HIGH=read) (will not be used; we will only write)
- Clock, enable
- Bit 0 (Not used in 4-bit)
- Bit 1 (Not used in 4-bit)
- Bit 2 (Not used in 4-bit)
- Bit 3 (Not used in 4-bit)
- Bit 4
- Bit 5
- Bit 6
- Bit 7
- Backlight LED Anode (+)
- Backlight LED Cathode (-)

Before we connect the display to the Raspberry Pi, we need to check if there is a resistor present leading to the backlight. If there is none, or you are not sure if there is one, a 1 Kohm resistor can be used to make sure there is resistance. The previous list follows the most common pin number scheme of the display. Some displays have the pin numbers 15 and 16 swapped. This changes the order of the pins. When you have the pin connections at the bottom, the pin numbering from left to right will be: 15, 16, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1. This is often shown on the board that the LCD is attached to, where you will be able to read which connection is pin number 1, and if 15 and 16 are swapped, it will be printed on the board.

These displays need 5V to operate and to use the backlight. It is no problem to use these if we are only writing to the display, as we will be doing, there is no risk. To make sure the display also understands we only want to write, we set the R/W pin to low by connecting it to ground. We will use a 10 Kohm potentiometer (variable resistor) to be able to show the text on the display.

Coming up is a screenshot showing the connections made when using a breadboard using male to female jumper wires. The female part of every wire is connected on the Raspberry Pi and the male part in the breadboard. The full setup is available in the download. The following screenshot shows the LCD display upside-down to make it easier to link the pin numbers as they are now from left to right. When the display is placed correctly, the connections are placed on the bottom, and the pin numbers 1 to 16 are from right to left.

Coming up is a list showing which pin numbers on the Raspberry Pi connect to the pin numbers on the LCD. When we use the T-cobbler to connect the parts, the pin numbers will also be shown on the T-cobbler. Most of the lanes on breadboards are red and blue. If the colors are different, remember where you connect the 5V and the Ground connections. Make the following connections with the Raspberry Pi shut down and the power disconnected. Before we start to connect, it is wise to have the following schematic present so we can map the correct pin numbers. This is where close attention is needed because there different pin numbering schemes out there. This is the BCM pin numbering scheme and the GPIO numbering scheme. We will be following the GPIO pin numbering scheme from `PI4J/WiringPi` as the libraries we will be using follow this scheme. Any pin number reference will have the numbers printed in bold:



The previous image is has the Raspberry Pi with the pins facing you and situated on the top-right, the USB connectors on the bottom, and the HDMI connector pointing left.

Let's connect the display to the Raspberry Pi 3.

1. Connect the 5V (pin 2) on the Raspberry Pi to the red lane on the breadboard.
2. Connect the ground (GRND, pin 6) from the Raspberry Pi to the blue lane on the breadboard.
3. Connect pins 1 and 16 on the LCD to ground.
4. Connect pins 2 and 15 on the LCD to the 5V lane. If you aren't sure that 15 has a resistor, put a 10 Kohm in between (not shown on the schematic).

5. Connect pin 5 on the LCD to ground (R/W pin).
6. Depending on your used potentiometer, connect the left pin to ground, the right pin to 5V, and the center pin to LCD pin 3 (Contrast).
7. Connect pin 4 on the LCD to GPIO pin 01 on the Raspberry Pi.
8. Connect pin 6 on the LCD to GPIO pin 04 on the Raspberry Pi.
9. Connect pin 11 on the LCD to GPIO pin 26 on the Raspberry Pi.
10. Connect pin 12 on the LCD to GPIO pin 27 on the Raspberry Pi.
11. Connect pin 13 on the LCD to GPIO pin 28 on the Raspberry Pi.
12. Connect pin 14 on the LCD to GPIO pin 29 on the Raspberry Pi.

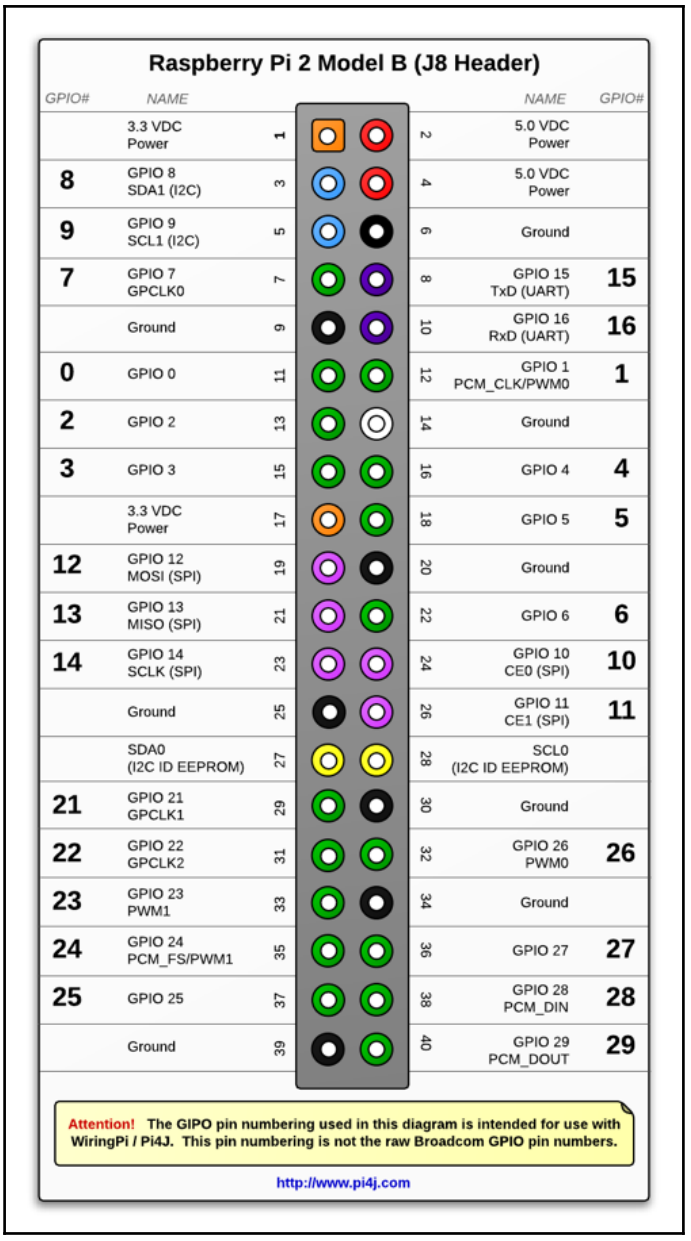
We now have the 16x2 Character LCD connected to the Raspberry Pi 3. It is now time to display data on this display. Turn on the Raspberry Pi by connecting the power. We will see the LCD display lit up.

Showing data on the HD44780-compatible display

Now that we have connected the LCD to the Raspberry Pi, it is time to actually start with the code and display some data. With the project loaded in NetBeans, open the `Chapter2.java` class, which lives in the, `chapter2` package. Let's take a look at the `main(String args[])` method.

We see all the methods are commented out. We will be uncommenting these methods throughout the book, so each step will get more obvious, and we will be able to follow and explain all the interactions between the components and the software.

Uncomment the `runLcdExample();` method and press the **Run** (with the green arrow) button in the IDE. NetBeans will start to compile the application; upload it to the Raspberry Pi and run it. If we have set up the wiring correctly, we will see the following screenshot, but of course, with your local time. If you are not able to see the text, we need to adjust the contrast of the display with the potentiometer we attached to pin number 3. In the following screenshot, the potentiometer is on the bottom right. To make the adjustment, you need to turn the knob to the left or right to be able to show the text:


Raspberry Pi 2 Model B (J8 Header)
GPIO# NAME NAME GPIO#
3.3 VDC Power 1 2 5.0 VDC Power
8 GPIO 8 SDA1 (I2C) 3 4 5.0 VDC Power
9 GPIO 9 SCL1 (I2C) 5 6 Ground
7 GPIO 7 GPCLK0 7 8 GPIO 15 TxD (UART) 15
Ground 9 10 GPIO 16 RxD (UART) 16
0 GPIO 0 11 12 GPIO 1 PCM_CLK/PWM0 1
2 GPIO 2 13 14 Ground
3 GPIO 3 15 16 GPIO 4 4
3.3 VDC Power 17 18 GPIO 5 5
12 GPIO 12 MOSI (SPI) 19 20 Ground
13 GPIO 13 MISO (SPI) 21 22 GPIO 6 6
14 GPIO 14 SCLK (SPI) 23 24 GPIO 10 CE0 (SPI) 10
Ground 25 26 GPIO 11 CE1 (SPI) 11
SDA0 (I2C ID EEPROM) 27 28 SCL0 (I2C ID EEPROM)
21 GPIO 21 GPCLK1 29 30 Ground
22 GPIO 22 GPCLK2 31 32 GPIO 26 PWM0 26
23 GPIO 23 PWM1 33 34 Ground
24 GPIO 24 PCM_FS/PWM1 35 36 GPIO 27 27
25 GPIO 25 37 38 GPIO 28 PCM_DIN 28
Ground 39 40 GPIO 29 PCM_DOUT 29
Attention! The GIPO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.
http://www.pi4j.com

Let's take a look at what has happened to be able to show the text on the display.

In our project, we have a `wrapper` class, which makes it easier to interact with the LCD display. This `wrapper` class has the methods of communicating with the LCD display wrapped in convenience methods so our development will be quicker and easier. Before we continue, we take a look at the method running in the example.

To quickly jump to methods within NetBeans, press command key on Mac and Ctrl key on Windows and Linux and click with your mouse on the method. In the book, it is mentioned as following a method or class.

When looking inside the `runLcdExample()` method, we can see a class being initialized with the name `LcdExample();`. Follow the `LcdExample();` class; we will now be taken to a new window with the contents of this class. This class is very small and it will show the use of a `wrapper` class that performs the actual actions. Scroll down to the constructor where we see that it is calling the static `getInstance()` method. Later, we will take a look at this static method where the LCD code is being initialized in order to prepare the code to run the LCD. Scroll down further to the `runExample()` method, where the example is writing the data to the LCD display. Let's take a look at this:

```
/**
 * Run the LCD example.
 */
public final void runExample(){
    /// Clear the display
    handler.clear();
    /// Cursor to home position (0,0)
    handler.setHome();
    /// Write to the first line.
    handler.write("-- RASPI3JAVA --");
    /// Create a time format for output
    SimpleDateFormat formatter = new
    SimpleDateFormat("HH:mm:ss");
    /// Sets the cursor on the second line at the first
     position.
    handler.setCursor(1, 0);
    /// Write the current time in the set format.
    handler.write("--- " + formatter.format(new Date()) +
    " ---");
}
```

In this example, we see convenience methods used to quickly be able to perform the actions in this example. First, clear the display with the `handler.clear();` method, which is a method within the parent class. With the `handler.sethome();` method, we position the cursor of the LCD display to its home position, which is `row 0` and `column 0`. These are 0 because the row and column indexes are zero-based. So, the first line is identified with `row 0` and the first column is identified with `column 0`. After we have set the cursor to its home position, we write out the first text with `handler.write("-- RASPI4JAVA --");`. As you can see in the previous screenshot, this is on the first line of the display. We now set the format to display the time and move the cursor to the second line in the first column with `handler.setCursor(1, 0);`. Now it's time to put the current time on the display, so we write this to the LCD with `handler.write("--- " + formatter.format(new Date()) + " ---");`. All these convenience methods called are present in the parent class. Having these methods will come in handy when we continue with the project. Let's take a look at what has actually happened, step-by-step. Scroll back up to `public final class LcdExample {` and follow the `LcdHandler.getInstance();` method by clicking on the `LcdHandler` part and holding command/Ctrl key. Now that we have the class open where all the magic happens, let's take a look and break it down:

```
package chapter2.lcd;

import com.pi4j.component.lcd.impl.GpioLcdDisplay;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.RaspiPin;

/**
 * The main LCD handler class.
 * @author John Sirach <john.sirach@pidome.org>
 */
public class LcdHandler {

    /**
     * The amount of rows on this display.
     */
    private final static int LCD_ROWS = 2;
    /**
     * The amount of columns per row.
     */
    private final static int LCD_COLUMNS = 16;

    /**
     * GPIO controller.
     */
    final GpioController gpio = GpioFactory.getInstance();
```

```
/**
 * The LCD handle we will be using to communicate with
the display.
 */
final GpioLcdDisplay lcdHandle;

/**
 * For singleton use.
 */
private static LcdHandler handler;

/**
 * LcdHandler constructor.
 * @throws LcdSetupException When init of the LCD
fails.
 */
private LcdHandler() throws LcdSetupException {
    // initialize LCD
    lcdHandle = new GpioLcdDisplay(LCD_ROWS,          // Rows
                                   LCD_COLUMNS,       // Columns
                                   RaspiPin.GPIO_01,  // RS
                                   RaspiPin.GPIO_04,  // Enable
                                   RaspiPin.GPIO_26,  // Bit 1
                                   RaspiPin.GPIO_27,  // Bit 2
                                   RaspiPin.GPIO_28,  // Bit 3
                                   RaspiPin.GPIO_29); // Bit 4
}

/**
 * We use an singleton because we need to make sure
   only one instance lives.
 * @return The lcd handler.
 * @throws chapter2.lcd.LcdSetupException When the
   setup of the LCD display fails.
 */
public static LcdHandler getInstance() throws
LcdSetupException {
    if(handler == null){
        handler = new LcdHandler();
    }
    return handler;
}

/**
 * Clears the LCD display.
 */
public final void clear(){
    lcdHandle.clear();
```

```
    }

    /**
     * Places the cursor at row 0 and column 0.
     */
    public final void setHome(){
        lcdHandle.setCursorHome();
    }

    /**
     * Sets the cursor at a specific position.
     * The row count starts at 0. This means the first
       line is row 0, the second line is row 1.
     * The column count starts at 0, this means the first
       column is 0, the second column is 1.
     * @param row Sets the cursor on the defined row.
     * @param column Sets the cursor on the defined
       column.
     */
    public final void setCursor(int row, int column){
        lcdHandle.setCursorPosition(row, column);
    }

    /**
     * Write a text on the cursor position.
     * @param text The text to write to the LCD.
     */
    public final void write(String text){
        lcdHandle.write(text);
    }

}
```

This class has been set up as a `singleton` class. The main reason for this is we want to make sure that only one instance of the LCD instance is available. We are setting some constants identifying the LCD we are interfacing with. We have two rows and 16 columns per row. In our project, we are communicating with 4 bits to the display, so we use only four pin parameters. The main reason for this is we otherwise would need to use four more pins on the Raspberry Pi that we could be using for other purposes. To make this class able to provide usable convenience methods, we need a variable that we can re-use within them. We first need to get access to the controller of the GPIO pins. This is done with `final GpioController gpio = GpioFactory.getInstance(); .`

This controller provides us with all the methods needed to be able to communicate with the GPIO pins on the Raspberry Pi. We have a class accessible variable called `lcdHandle`. This variable holds the reference to the LCD interface being instantiated in the constructor and is identified by `final GpioLcdDisplay lcdHandle;`. In the constructor of the class, we initialize the LCD so we are able to communicate with it. The following code shows the initialization:

```
// initialize LCD
lcdHandle = new GpioLcdDisplay(LCD_ROWS,          // Rows
                               LCD_COLUMNS,       // Columns
                               RaspiPin.GPIO_01,  // RS
                               RaspiPin.GPIO_04,  // Enable
                               RaspiPin.GPIO_26,  // Bit 1
                               RaspiPin.GPIO_27,  // Bit 2
                               RaspiPin.GPIO_28,  // Bit 3
                               RaspiPin.GPIO_29); // Bit 4
```

The LCD initialization begins with the LCD configuration using `LCD_ROWS` (`2`), `LCD_COLUMNS` (`16`) we have set as constants on top in the class. Remember we have initialized the `Pi4J` library to use the GPIO pin numbering scheme in the `Main` class? Because we did this, the numbering of the GPIO pins in this convenience class must also follow the GPIO numbering scheme. The `Pi4J` has convenience pin numbering constants available, which are located in the `RaspiPin` class. These convenience constants all start with `GPIO_` and are followed by the pin number. The pin order to initialize is first the RS pin (4 on the LCD display) we connected to GPIO pin number 01. This combination results in `RaspiPin.GPIO_18`, which returns the correct information for using GPIO pin number 18. Now, to break down the convenience methods, we have a few of them defined, which are `clear()`, `setHome()`, `setCursor(int row, int column)`, and `write(String text)`. The `clear()` method wipes the screen and removes all characters.
The `setHome();` method moves the cursor to `0,0`.

The `setCursor` method locates the cursor of the LCD to the position passed in the method minus 1. As the LCD display is zero-based, the first line and first character are located at `0,0`. This means when we use this convenience method and want the character cursor to be placed on the second line at the fifth position; we would use `setCursor(1, 4);` to do so. This method also wraps the original `lcdHandle.lcdPosition (column, row);` method. The `write` method is the method used to write characters to the display. It is important that this method is called after setting the location of the cursor. This will make sure the characters are written at the correct location.

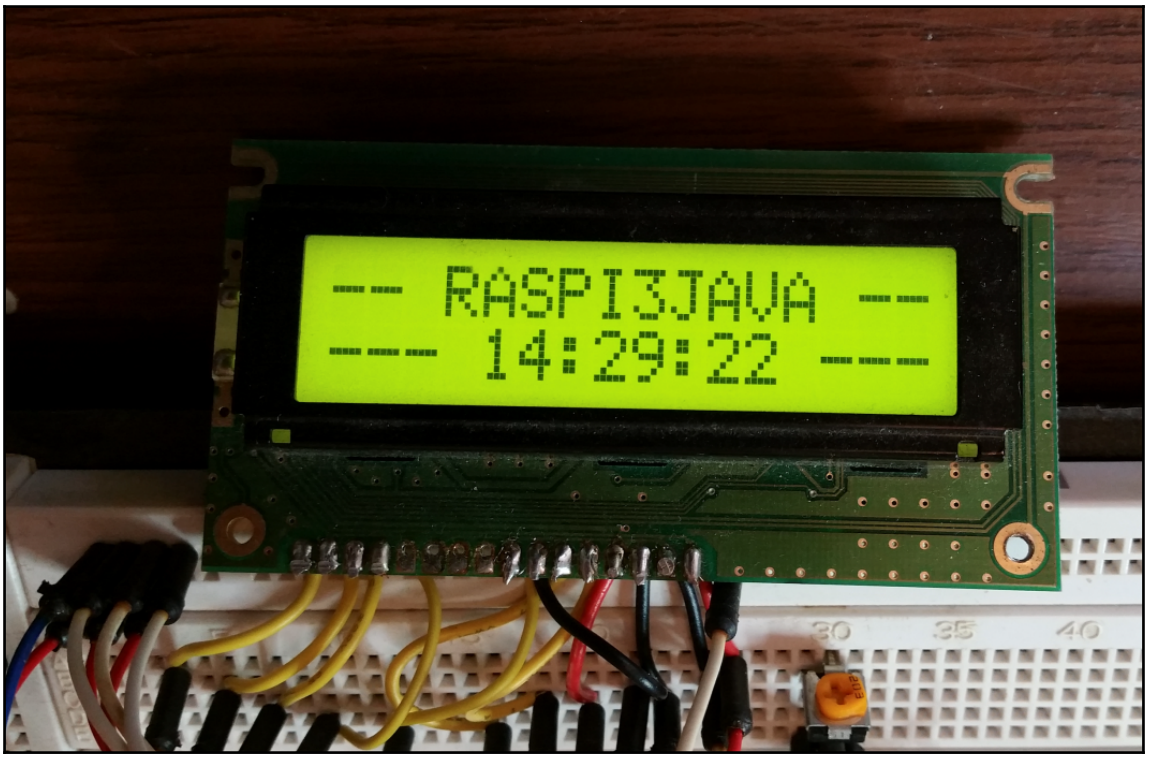
In the rest of the project, we will be using the `LcdHandler` wrapper class to show characters on the display. Let's close the `LcdHandler` class by pressing the **x** on the `LcdHandler` class tab. When we click the **LcdExample** tab, if not already in front, or if closed, open it from the `chapter2.lcd` package. We can start making some modifications so we will say `hello` to our self.

Scroll down to the `runSecondExample()` method. In this method, on the first line we say `hello`. Complement this code by setting the cursor to the second line and writing down our name. You can use the breakdown we just walked through to set the cursor position and write to the display. Press the **Run** button in the editor to see it displaying on the LCD.

Adding the light-dependent resistor to the setup

As mentioned earlier, the Raspberry Pi is unable to read analog values. We are going to add the resistor using a RC circuit, as explained earlier. Here, we need the LDR, capacitor, and a 4.7 Kohm fixed resistor. The fixed resistor is used to make sure that when the LDR is completely saturated, which means that there is no resistance anymore, we won't fry our Raspberry Pi. An extra thing we need to keep in mind is that the Raspberry Pi is a 3.3V device. This means that in this schema, we will be using the 3.3V output because we will be reading the input on the Raspberry Pi pin, which cannot be higher than 3.3V.

Here is an image that shows how to attach this RC circuit to the Raspberry Pi:



Shut down the Raspberry Pi and disconnect the power. Let's first take a quick look at the capacitor. The one we are using is an electrolytic capacitor, which has a positive and a negative pin. The negative pin is easily identified, as when you take a close look it is marked with a line with a different color from the capacitor. This line has the - sign on it. This way we know this side is the ground side. When we follow the wires to connect the previous schematic from 5V to the Raspberry Pi, we connect it as follows:

1. Connect the 5V to one side of the 4.7 Kohm resistor.
2. Connect the other side of the resistor to the LDR.
3. Connect the other side of the LDR to the 1 μF capacitor.
4. Connect the negative side (marked with the - sign) to ground.
5. Connect the side of the LDR where the capacitor's + pin is to the Raspberry Pi's GPIO 05 pin.

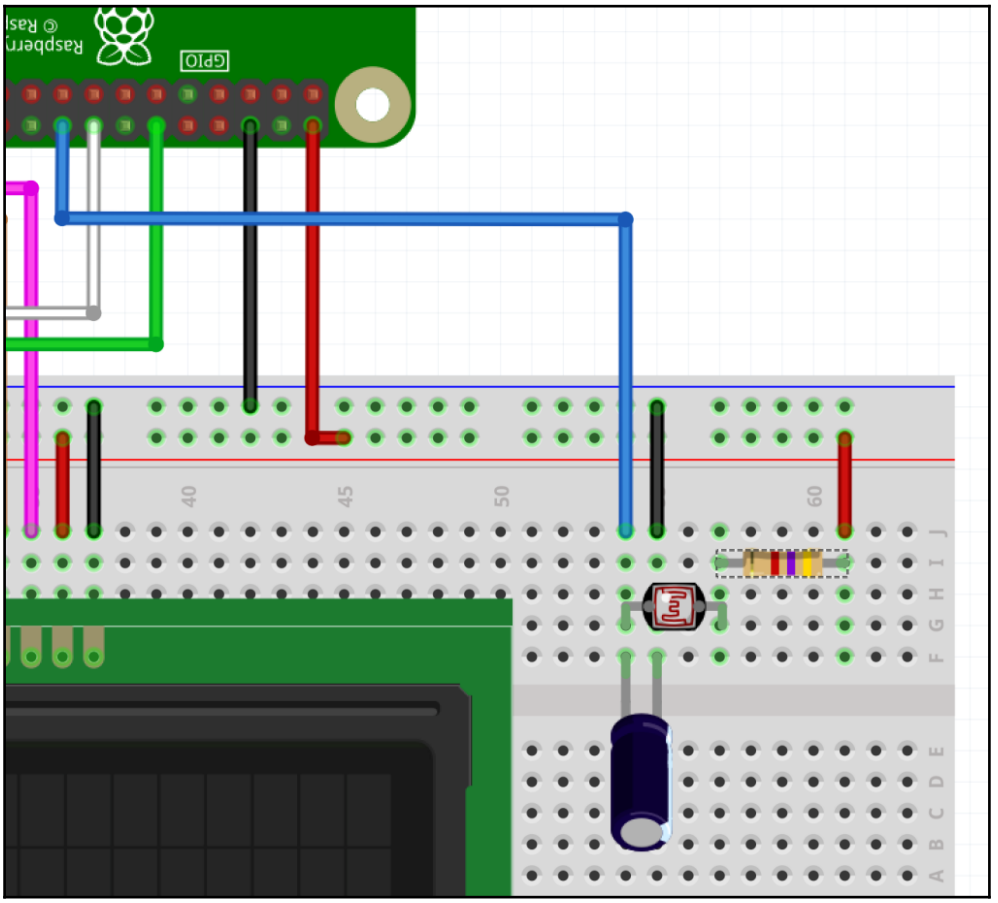
What we will be doing is measuring the time it takes before the Raspberry Pi detects a high state on GPIO 05. In this schematic, the following will happen. The current flows from the positive side through the 4.7 Kohm resistor and the LDR. When the Raspberry Pi GPIO pin 05 is set to pin mode INPUT, it will block the current to flow through this PIN. This results in the current flowing into the capacitor and while this is happening, the voltage will get higher depending on the resistance of the LDR. The higher this is (less light), the longer it will take for the capacitor to be filled up causing the voltage to rise slowly. When the voltage is high enough for the Raspberry Pi pin to detect a high, we will do some calculations to see how long this took. When this calculation is done, we change the PIN mode on the Raspberry Pi to be output and set the pin state to LOW. This will cause the capacitor to drain its build up voltage and become empty. Then we restart the process by setting the PIN mode to INPUT again. This will cause the Raspberry Pi to detect a LOW state, as there is no or not enough voltage, and calculate again how long it took to reach the high state again.

Reading and displaying the values from the LDR

Doing this needs some experimentation on the values we want to have, especially because everyone's definition of *It is dark* is different. Remember, the longer it takes for the pin to detect the high state, the darker it is.

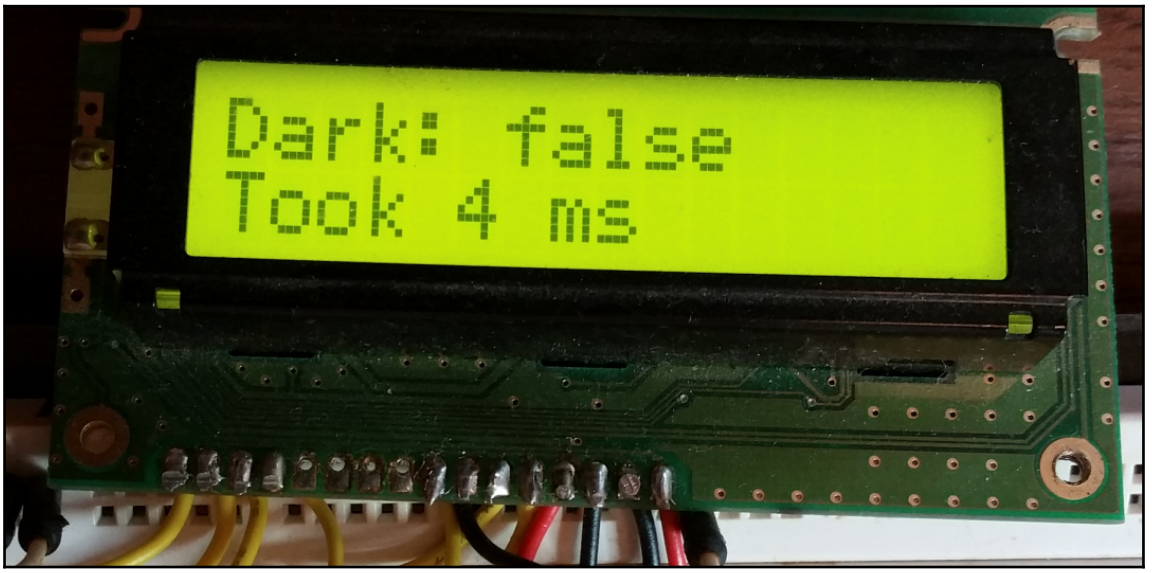
Close all the tabs except the `Main` class, where we will be uncommenting the next example to run. Scroll to the `main` function where we will be commenting the `runLcdExample();` and uncomment the `runLdrExample();` method. This next method runs the example of how to measure light intensity. When the LDR is connected like in the previous directions, you are able to run the application again by pressing the **Run** button in the button bar. Take a look at the LCD display where we see different text appearing depending on the light intensity surrounding the LDR.

When there is a lot of light, the display could show information as shown in the following screenshot:



In this picture, we see that it took only 4 milliseconds for the capacitor to fill up until it reached the point where the Raspberry Pi pin detects a high value. As the current threshold in the class, which we will be looking at later, has been set to 40 milliseconds, the result is that it isn't dark.

When we cover the LDR, or make the surroundings a lot darker and we run the same application again, we see a result as shown in the following picture:



We now see that it takes at least 41 milliseconds to detect the capacitor to fill up enough to trigger a high on the Raspberry Pi pin. We say at least because it is possible it is so dark, the voltage will never be high enough to trigger a HIGH. As long as there is no high detected, the application will not be able to provide the information that it is dark. In the code we will be looking at, there is a statement that says that if it takes longer than the set threshold, we can be sure it is dark.

Let's scroll down in our `Main` class until we are at the `runLdrExample()` method. Within this method, the class to read the light intensity is instantiated. Follow the `LdrReader()` class by holding the command/Ctrl key and primary clicking on it. We are now in the class responsible for manipulating the Raspberry Pi pin. Let's take a closer look at it from top to bottom.

We start with registering the controller in this class. This is possible as the controller used in multiple classes is retrieved from a singleton within the Pi4J libraries, making sure there is only one controller available. We assign the controller with `private final GpioController gpioController = GpioFactory.getInstance();`. With this instance, we can take control of the pins. Before we start with the pin, we define it first by setting a variable to GPIO 05 with `private final Pin gpioPin = RaspiPin.GPIO_05;`. Now that we have the controller available and the pin defined, we can create a `Pin` object. We are then able to manipulate the pin physically through Java.

We define the `Pin` object globally with `private final GpioPinDigitalOutput ldrPin;` and create a fresh object within the constructor. But before we look at this, we need to set the variable that is the threshold of our definition of dark. By default, it is set at 40 milliseconds, which in my case, with the LDRs I have available, is around sunset level. Your case might be different so this is the value we will be playing with. This threshold is defined as `private final long darkThreshold = 40;` which is 40 milliseconds.

Now that we have our variables in place, we can construct the `Pin` object in the constructor with `ldrPin = gpioController.provisionDigitalOutputPin(gpioPin);`. The `ldrPin` object gives us methods to put the pin in input or output mode and in a low or high state. We will be using the input and output modes with the pin set to the low state.

Let's scroll down to the `pinIsHigh()` method, where we check if the pin is in a high state or not. We skip the `singleDarkDetect` method to be viewed last. The `pinIsHigh()` method returns a Boolean whether the pin we have defined is in a low or high state by executing the `ldrPin.isState(PinState.HIGH);` method return. This method literally only checks the physical state of the pin. When the pin is in low state, this method will return `false`. Let's scroll down further to the `setPinDrain();` method . Although this method name is a little misleading, it explains what we are trying to achieve. See the following method:

```
/**
 * Change the pin to digital out and to low.
 * This causes to connect this pin to ground.
 */
private void setPinDrain(){
    ldrPin.setMode(PinMode.DIGITAL_OUTPUT);
    ldrPin.setState(PinState.LOW);
    try {
        //// Give the capacitor some time to get drained.
        Thread.sleep(500);
    } catch (InterruptedException ex) {
        System.err.println("Not waiting 500 ms to drain,
        result not guaranteed.");
```

```
            }
        }
```

This method changes the pin mode to be a digital out pin using `ldrPin.setMode(PinMode.DIGITAL_OUTPUT);`. If the pin in this mode is set to high it will output 3.3V. We want to set the state of the pin to low so it will connect to ground which will give the capacitor the opportunity to discharge. We set it to low using `ldrPin.setState(PinState.LOW);`. We also want to make sure the capacitor is fully drained so we wait for 500 milliseconds before we continue. This is quite long, but if you have chosen to use a bigger capacitor, it's better to be safe and make sure it is quite empty. Scroll down further until we reach the `setPinToDetect()` method . This method name can also be misleading as we are changing the pin mode but it also explains our purpose. We want to change the pin mode to be able to detect a high state by setting the pin back to INPUT mode with `ldrPin.setMode(PinMode.DIGITAL_INPUT);`. This pin state is able to detect when the voltage is at a certain level so it detects a high or a low state.

Now that we have covered the pin modes and states, we are able to use these and start measuring the environment light intensity. Scroll back up to the `singleDarkDetect(boolean singleOutput)` method, which has the following contents:

```
    public final boolean singleDarkDetect(boolean singleOutput){

        long measure = 0;
        boolean dark = false;

        setPinDrain();
        Date startMeasure = new Date();
        setPinToDetect();

        while(!pinIsHigh()){
            Date stopMeasure = new Date();
            measure = stopMeasure.getTime() -
            startMeasure.getTime();
            if(measure > darkThreshold){
                dark = true;
                break;
            }
        }

        if(singleOutput){
            try {
                LcdHandler lcdHandler =
                LcdHandler.getInstance();
                lcdHandler.clear();
                lcdHandler.setHome();
```

```
                lcdHandler.write("Dark: " + dark);
                lcdHandler.setCursor(1, 0);
                lcdHandler.write("Took " + measure + " ms");
            } catch (LcdSetupException ex) {
                System.err.println("Failed to write to LCD: "
                + ex.getMessage());
                System.out.println("It is dark: " + dark + ",
                took " + measure + " ms");
            }
        }
        return dark;

    }
```

Let's break down this method. We start with some default values, we set our `long measure = 0;` and we predefine that it's light enough by setting `boolean dark = false;`. We need these variables in a loop and want to output the values when the loop is done, so we need to predefine them. Our first action is to drain the capacitor, so we call the `setPinDrain();` method as this also contains the sleep method that waits a bit to make sure the capacitor is drained. When the method returns, we set the start time with `Date startMeasure = new Date();` so we can use this during a loop to measure the time it takes to detect the high state. After we set the start time, we change the pin status with `setPinToDetect();` so we are able to start detecting when the pin is in a high state. Now we need to wait for the pin to get high. So we create a while loop that runs as long as the pin is in a low state by continuously asking for the current pin state with `while(!pinIsHigh()){/* code */ }`. In this while loop, we instantiate a new `Date()` object, which is used in the calculation `measure = stopMeasure.getTime() - startMeasure.getTime();` to see if the pin state is still within the threshold for us to say it is light. When this measure value is more than the threshold, checked with `if(measure > darkThreshold){/* code */}`, we can say the surroundings are dark by setting `dark = true;`. So we have now detected that the measurement is higher, so we do not need to loop anymore because our threshold has already been passed. So, we `break;` out of the loop. In this way, we also have a failsafe in case the surroundings are very dark that causes the pin to never reach a high state. If we did not break from the loop, this loop would run forever until a high state is encountered.

Via the example method we are using to call `singleDarkDetect`, we have set the `singleOutput` parameter to `true`. Because of this, we will be able to see the result being displayed on the character display. Here we request the LCD `wrapper` class via the static singleton `LcdHandler.getInstance();` method. We are using the same methods as described earlier to show data on the LCD display, albeit now with our measurement results.

We now have the ability to write data to a display and are able to roughly measure the light intensity using an RC circuit. We have gone quite in-depth with these two subjects. Let's make the next section a bit lighter, and turn on a light using a relay.

Using digital out to switch and display a relay status

For switching a relay, we only need to use one pin on the Raspberry Pi and have its mode set to OUTPUT and turn this high or low to open or close a relay. Most relays need 5V to be able to switch, so we will be using a 2N3904 NPN transistor. With this transistor, we are able to switch a 5V lane with 3.3V so we can switch a relay.

A word of caution: Switching with mains voltages can be very dangerous. Only do this when you are absolutely sure what you are doing. Also, when beginning with relays, make sure you use a mechanical relay as these are able to switch any load up to the maximum rated on the relay.

In our schematic, we will only be switching the relay. This is done for safety reasons (please read **A word of caution**). When you are more experienced, loads can be added to the relay or a solid-state relay, which suits AC loads better. I would advise that you start with a 9V battery-supplied light bulb. Our schematic will be built such that the Raspberry Pi will need little effort to switch the relay. We will be using a BC546B NPN transistor to switch the relay's ground connector. This transistor, when not powered by the Raspberry Pi (pin is low), disconnects the ground from the relay from the ground attached to the Raspberry Pi. When we set the Raspberry Pi pin to high, the transistor will connect the ground, making the relay do its work and close the connection on the switching side powering the attached peripheral.

This particular relay is a normally open relay, which means the connection on the side to be switched is open, not connected. It has a power and a signal pin where the power pin is used to power the relay, the signal pin is used to let the relay know it can be powered which closes the connection on the switching side powering any attached peripheral. As long as the ground is disconnected, the state of the switch remains open. We will use the following schematic:

The yellow wire is connected to `GPIO_6` according to the `Pi4J` numbering scheme, which is pin number 22. Make sure the Raspberry Pi is shut down and is unpowered. The setup is connected as follows:

1. Connect pin 22 on the Raspberry Pi to the breadboard to the 10 Kohm resistor.
2. At the other side of the resistor, connect an extra 10 Kohm resistor to ground. This is a failsafe in case the pin on the Raspberry Pi is floating and starts switching the relay.
3. Connect the same other side of the resistor to the base of the transistor. For the BC546B, it is the middle pin.
4. Connect the left pin, the collector, of the transistor to the ground pin of the relay.
5. Connect the right pin to the ground lane on the breadboard. We have now finished the first part of switching the relay.
6. Connect the relay's power and signal pins with the 5V lane.

With this setup, it is very important not to attach anything to the relay yet because we want to test it first. To be able to do this, we need to go to the code in the `Main` class. If the previous `runLdrExample();` method is not commented out yet, we will do this now. Now, you can uncomment the `relayExample();` method, which will turn on and off the relay, and then stop the application. When we have uncommented the `relayExample();` method and run the application and all the connections are correctly set up, we will hear the relay switch to a closed state and after 2 seconds back to a closed state. If this is not the case, check the wiring. While the application is running, the state of the relay will be shown on the LCD display.

In the `Main` class, scroll down up to the `relayExample()` method. In this method, we instantiate the `RelaySwitch` class that runs the code to be able to switch the relay. Follow the `RelaySwitch` class by holding command/Ctrl key and click the mouse button. We now have the `RelaySwitch` class opened. Let's break down this class; we will see a couple of familiar methods passing by.

At the top of the class, we see the familiar variables being defined and having objects assigned. Again, we need the `GpioController` class to be able to instantiate a `GpioPinDigitalOutput`. This time we define our `Pin` as `GPIO_6`. Scroll down to the constructor. This time we do things a little bit differently, as we would like to set the pin in a predefined state before we change any states. Take a look at the following code:

```
/**
 * The constructor.
 * The constructor sets the pin mode to output and sets the state to
low.
 */
public RelaySwitch(){
    relayPin =
    gpioController.provisionDigitalOutputPin(gpioPin);
    relayPin.setMode(PinMode.DIGITAL_OUTPUT);
    relayPin.setState(PinState.LOW);

    LcdHandler localLcd;
    try {
        localLcd = LcdHandler.getInstance();
    } catch (LcdSetupException ex) {
        System.err.println("No LCD output available: " +
        ex.getMessage());
        localLcd = null;
    }
    lcd = localLcd;
}
```

As in the `LdrReader` class, we instantiate the pin as a provisioned output pin with `relayPin = gpioController.provisionDigitalOutputPin(gpioPin);` method, only this time we will not change this pin to become an input as we only want to set this pin in a high or low state. Because we want to make sure it is used as an output pin, we call the `relayPin.setMode(PinMode.DIGITAL_OUTPUT);` method and set it to a low state, as we want the relay to be open with `relayPin.setState(PinState.LOW);`. As we have the LCD available, let's use it again. In the `lcd` class member definition we have declared it final, as we will be using `lambda` in this example. We need to assign an object/null value via the constructor by first trying to get a method local instance of the `LcdHandler` by calling `localLcd = LcdHandler.getInstance();`. If this fails, we assign the value `null` to the local variable so we are able to assign a value to the final `lcd` member. With our members set, let's take a look at the method executing our example named `runExample()`:

```
public final void runExample(){

    if (lcd!=null) { lcd.clear(); lcd.setHome(); }

    relayPin.setState(PinState.HIGH);

    if (lcd!=null) lcd.write("Relay closed");

    ScheduledExecutorService closeRelay =
    Executors.newSingleThreadScheduledExecutor();
    ScheduledFuture<Boolean> future =
   closeRelay.schedule(() -> {
        relayPin.setState(PinState.LOW);
        if (lcd!=null){ lcd.setCursor(0, 0);
        lcd.write("Relay open  "); }
        return true;
    }, 2, TimeUnit.SECONDS);
    try {
        future.get(3, TimeUnit.SECONDS);
    } catch (InterruptedException | ExecutionException |
       TimeoutException ex) {
        System.err.println("An execution error occured: "
       + ex.getMessage() + " try to put the pin to low.");
        relayPin.setState(PinState.LOW);
        if (lcd!=null){
            lcd.setCursor(0, 0);
            lcd.write("Relay open  ");
            lcd.setCursor(1, 0);
            lcd.write("from error");
        }
    }
    closeRelay.shutdownNow();
}
```

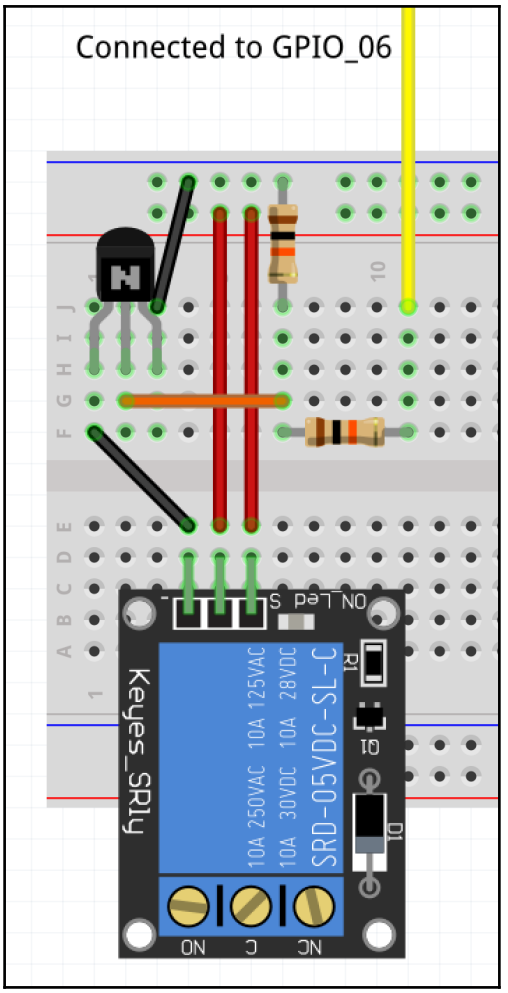
In this method, we will skip the LcdHandler methods as they have been explained in the previous two sections. The first interesting method is `relayPin.setState(PinState.HIGH);` which sets the pin mode to high. With the pin set to high, the transistor is getting a voltage on the base that closes the circuit between the relay and the ground lane. This allows the power and the signal pins to be connected, and as a result closes the switch circuit in the relay. Now we instantiate a scheduled executor because we want the pin set to low after 2 seconds with `ScheduledExecutorService closeRelay = Executors.newSingleThreadScheduledExecutor();`. Now that we have the executor, we can schedule our one shot runnable in this scheduled thread executor. We submit a lambda constructed `relayPin.setState(PinState.LOW);` method which sets the pin to the low state. With the pin low, we break the circuit between the relay and the ground plane, disconnecting the power and the signal lines. The relay will now be open again. We also want to make sure the pin is set to low again, so we have submitted the `Runnable` with a `Future` returned so we are able to cancel the operation after 3 seconds with `future.get(3, TimeUnit.SECONDS);` method. If after 3 seconds the relay is not closed, we still close it by catching the thrown exceptions.

We now have a couple of functions available that we are able to combine. We have the display to be able to display statuses, we can detect light intensity, and we are able to switch using a relay. This means we can already start automating things. Let's get some results from our hard work.

Automatic switch based on environment lighting

In this section, we will be combining the built hardware components to automate the switching of the relay based on the environment lighting, including printing states to the LCD. Let's close all the code tabs in the editor and leave only the `Main` class open. If not already done, comment the `relayExample();` method and uncomment the `lightDependentSwitching()` method. This last method will instantiate the `LdrHandler` class and pass on a `RelaySwitch` object. If you already have been playing with the `darkThreshold` member value in the `LdrHandler` class, you can press the Run application button and the application should start switching automatically and update the LCD statuses. When we play a little bit with the environment lighting by putting our hand over the LDR, the relay should close, and when light is reaching the LDR again, the relay should open.

We now have this application continuously running because of live threads. To stop the execution of the application, we can use two methods. At the bottom of the NetBeans editor, we can see the progress bar of the application currently running, which looks like the following screenshot:



By clicking on the small cross on the right after the name and progress bar of the running application, we can stop the current running process. The second method is by pressing the **Run** menu option in the menu bar of NetBeans, with the bottom option being `Stop Build/Run: Chapter2 (run-remote)`, as shown in the following screenshot:

Let's stop the current running `Chapter2 (run-remote)` process and take a look at the code we just executed and stopped.

When scrolling down in the `Main` class, we see the `lightDependentSwitching();` method, which runs the automatic switching. We have already discussed the classes being instantiated in the first two lines of this method. On the third line, the `ldrReader.runAutomaticLightSwitching(relaySwitch);` method runs the code. We can jump immediately to the correct code when we hold command/control and click on the `runAutomaticLightSwitching(relaySwitch)` method. We are now in the `LdrReader.java` class file and looking at the executing method.

Again we are requesting the `LcdHandler` using the `getInstance()` method and put it in the final `lcdHandler` variable. After getting the LCD handler, we instantiate a `ScheduledExecutorService` containing two threads with `intensityCheckScheduler = Executors.newScheduledThreadPool(2);`. These two threads are getting two Runnables submitted. The first one is used to display the current time on the display, which is being refreshed every minute with the following code:

```
final SimpleDateFormat formatter = new SimpleDateFormat("HH:mm");
intensityCheckScheduler.scheduleAtFixedRate(() -> {
    writeLcd(lcdHandler, 0, 0, formatter.format(new Date()));
}, 0, 60, TimeUnit.SECONDS);
```

We define our time format with a new `SimpleDateFormat` instance so we can display it nicely on our display. A small convenience method is present in this same class, which we will be looking at later on. This method, which is also called in this `lambda` function, writes the data to the LCD. This `Runnable lambda` is executed every second, starting from the zeroth second, which are the second, third, and fourth parameters at the moment we schedule our time writer.

The second `Runnable lambda` contains the code that is responsible for checking if the current environment is dark or not and sets the relay state to open or closed:

```
intensityCheckScheduler.scheduleAtFixedRate(() -> {
    if(singleDarkDetect(false) == true){
        relaySwitch.closeRelay();
        writeLcd(lcdHandler, 0, 13, "On ");
    } else {
        relaySwitch.openRelay();
        writeLcd(lcdHandler, 0, 13, "Off");
    }
}, 0, 10, TimeUnit.SECONDS);
```

Again, we schedule this at a fixed rate starting from the zeroth second and repeat every 10 seconds. We use the same method as a couple of sections back, where we looked at the code on how to detect the environment lighting, which is `singleDarkDetect(false)`, only this time we do not want to write to the display so the parameter is set to false and we use the return value to set the relay state and write to the LCD. This time, we use methods to open and close the relay, which are `relaySwitch.closeRelay();` to close the circuit and `relaySwitch.openRelay();` to disconnect the circuit. The only line of code in these methods are `this.relayPin.setState(PinState.HIGH);` and `this.relayPin.setState(PinState.LOW);`.

The last method we will be looking at is the convenience method to write to the display. To make sure that the previously scheduled threads are not colliding while writing to the display, an extra executor service is instantiated. When we look at the code this will get more obvious:

```
/**
 * Write text on the lcd at the desired location.
 * @param lcdHandler The lcd handler.
 * @param row The row the cursor should be placed.
 * @param col The column the cursor should be placed.
 * @param text The text to write.
 */
private void writeLcd(final LcdHandler lcdHandler, final int row,
final int col, final String text){
    if(lcdHandler!=null){
        if(displayWriter == null){
            displayWriter = Executors.newSingleThreadExecutor();
        }
        displayWriter.submit(() -> {
            lcdHandler.setCursor(row, col);
            lcdHandler.write(text);
        });
    }
```

```
        }
```

The parameters used are explained in the code comments. What we are doing here is checking if the `lcdHandler` is not null because this is a possibility. We are checking if there is already an executor service running or not. If this is not the case, we instantiate a single thread executor with `Executors.newSingleThreadExecutor();`. The reason for selecting a single thread is this will give us almost a First In First Out executing environment for writing to the LCD, and it makes sure only one submitted Runnable is running at any time. Imagine what would happen if we did not do this. Text will start to appear in weird locations on the LCD.

Now that we are able to automatically switch the relay based on the brightness of the environment and display the status, the next and almost final step is that we only want to close the relays circuit when we are around. We are going to use Bluetooth to detect presence.

Using the Bluetooth chip on the Raspberry Pi

Since the Raspberry Pi 3, a Bluetooth chip that supports BLE 4.1 has been present on the board. Using Bluetooth from Java can be a challenging thing to do. The latest specification of Bluetooth integration is from the **JSR** (**Java Specification Request**)-82 from 2010, which is version 4, and is implemented in **J2ME** (**Java 2 Micro Edition**). Although this specification is implemented in J2ME, there are libraries available that provide this specification, although not completely, for J2SE. One of these libraries is called **BlueCove**.

BlueCove tries to comply with the JSR-82 specification published in 2010. Initial support from BlueCove is for non-ARM based devices. To be able to use Bluetooth on ARM-based devices such as the Raspberry Pi, we will need to create the necessary libraries ourselves or we can use pre-built libraries. The download already contains the libraries we need to interact with the Bluetooth chip, so it is not necessary for us to build them. What we will do in this section is initialize the Bluetooth chip, search for devices close by, and show them in the console. After this, we will put these devices in a map as a set of devices we allow to interact with the Raspberry Pi. This part is all code so there is no need for extra hardware.

Bluetooth device discovery

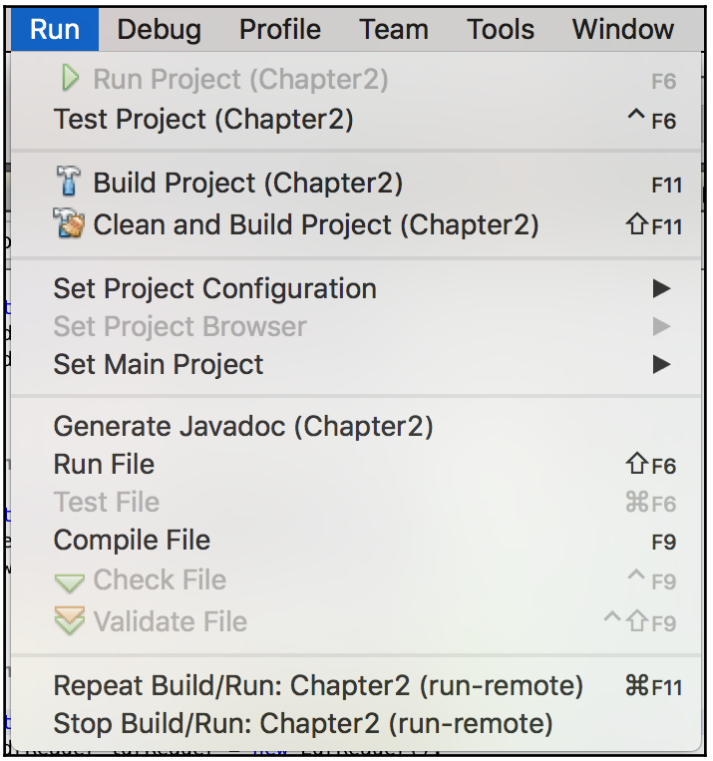
We will be only using the device discovery feature as this is the most stable part in the libraries on the Raspberry Pi. Also, there is only one mode available, which I will explain later. Let's close all tabs except for the `Main` class.

In the `main` method, have all the methods commented except for the `runBluetoothDetectionExample();` method. Follow this method, which causes us to scroll down to the contents of this method:

```
/**
 * Runs the Bluetooth device discovery example.
 */
private static void runBluetoothDetectionExample(){
    BluetoothDevices example = new BluetoothDevices();
    example.init();
    try {
        System.out.println("Starting device search,
    please wait.");
        example.runExample();
        System.out.println("Search done");
    } catch (DiscoveryFailedException ex) {
        System.err.println(ex.getMessage() + " because
   of " + ex.getCause().getMessage());
    }
    example.destroy();
}
```

This code makes use of our main `Bluetooth wrapper` class for interacting with the final `Bluetooth` class we will walk through later on. What we first do is initialize the `Bluetooth` class with `BluetoothDevices example = new BluetoothDevices();`. This gives us an instance we can initialize with `example.init();`. We do this because we are attaching listeners, and we do not want to do this from within the constructor. When the initialization has been done, we are starting the device discovery with `example.runExample();`. This method will show the devices found. This is wrapped within a try/catch because it is possible the Bluetooth chip on the Raspberry Pi is turned off, for example. When discovery has ended, we call the `example.destroy();` method, which causes the `wrapper` class to detach any listeners. Turn on the Bluetooth on your phone and make it visible for all devices and run the Java application.

The output in NetBeans should look a bit like the following screenshot:



As we can see in the screenshot, when the discovery starts, we are notified about the `BlueCove` library being active with the Linux bluez Bluetooth stack. When we run the Java application, we notice it looks like it is hanging and not responding. This is not the case, as it is performing a search. Every device found will be printed out to the console with the Bluetooth address and the device's name. It is possible the device's name cannot be requested if a remote device does not allow this. When the search is done, we will be notified about this and the application ends. If in your output your device is not showing, run the application a couple of times more until it shows. If it takes a couple of runs, which is possible, make a note of how many times you needed to run it to see your device. So, let's now take a look how the `BlueCove` library is used.

Follow the `BluetoothDevices()` method, which will open the `BluetoothDevices` class in a separate tab. This class is our `wrapper` class providing convenience methods for performing this search. We are going to walk through this class as if the example is running. As we see, we implement a listener called `DiscoveryServiceListener`. This listener class contains only one method, which looks same as `public void serviceDiscoveryDone();` We need to implement this class in our `wrapper` class, which has been done at the bottom. At the top of the class we define two fields. One is for our `Bluetooth service discovery` class, which is immediately instantiated with private final `DiscoveryService discovery = new DiscoveryService();`. This `discovery` field will be used to perform all our search actions on and to retrieve the list of found devices stored in the second field private `List<RemoteDevice> foundDevices = new ArrayList<>();`. When we scroll down, we find an empty constructor and the `init()` method. This method is used to add a listener to the discovery service with `discovery.addListener(this);`. The discovery service only accepts classes implementing the previously mentioned `DiscoveryServiceListener`. Let us first cover the `startSearch()` method by scrolling down. This is a simple method that starts the actual search for devices with `discovery.startSearch();`. This method is a blocking method, meaning the application will stop executing until this method has finished running. We are now running a method that executes the actual Bluetooth code. This code can throw two exceptions, which we will be looking at later on. We wrap these exceptions as a cause in a single new exception called `DiscoveryFailedException` and throw it, which will make the code using this method a bit cleaner. When we scroll back up to the `runExample()` method we see the code that caused us to see the output we saw earlier. We first start the search with `startSearch();`. As this blocks, we wait, and when done, we request a list of found devices we want to iterate through with `Iterator<RemoteDevice> iter = discovery.getDiscoveredDevices().iterator();`. With every device we walk through, we want to know the address and the device name and print it to the console. We first request a `RemoteDevice` from the `Iterator` with `iter.next();`. The address is requested with the `device.getBluetoothAddress()` method, which is always available. The name of the device will not always be available, as it may throw an `IOException`. This is mostly thrown when a device does not allow us to request its name, or the connection is not good enough. We request the name with `device.getFriendlyName(true)`. `true` is used as we always want to request the name in case a device name has changed. If an exception is thrown, we will still see the Bluetooth address and the reason why we were not able to get the name. Let's go to the actual Bluetooth code.

Scroll back up and follow the `DiscoveryService` class or open the `DiscoveryService.java` file, which is available in the `chapter2.bluetooth` package.

With the `DiscoveryService` class open, we immediately see classes imported from the `javax.bluetooth` package, which are coming from the `BlueCove` library and contain the `Bluetooth` classes. The `DiscoveryService` class needs to implement a listener because the actual discovery is done in a different thread that does callbacks to the class implementing `DiscoveryListener` and registered as the listener. On the top of this class, we define two fields. The first field is to put discovered devices in a list and is defined with `private final List<RemoteDevice> discoveredDevices = new ArrayList();` All discovered devices will be put in this field. The second field is used for registering the listeners that will be using this class and need to be notified when discovery has ended and is defined as `private final List<DiscoveryServiceListener> listeners = new ArrayList<>();`. The third field, named `searchLock`, is a plain `Object` that is used as a lock while the discovery is running. The following two methods, `addListener(DiscoveryServiceListener listener)` and `removeListener(DiscoveryServiceListener listener)`, are used to register and de-register listeners in this class. To make sure a class is only registered once, my personal preference is to always check if it is not already registered with `if(!listeners.contains(listener)){}`. When we scroll down further, we come across the method to return the found devices with `List<RemoteDevice> getDiscoveredDevices()`, which returns the `discoveredDevices` filed. We now come to the method executing the search:

```
    /**
     * Starts discovery search.
     * @throws InterruptedException When the search is
        interrupted
     * @throws BluetoothStateException When are not able
       to use the Discovery Agent.
     */
    public final void startSearch() throws
    InterruptedException, BluetoothStateException {
        synchronized(searchLock) {
            boolean started =
LocalDevice.getLocalDevice().getDiscoveryAgent().startInqu
   iry(DiscoveryAgent.GIAC, this);
            if (started) {
                searchLock.wait();
                notifyListeners();
            }
        }
    }
```

This is the method that is capable of throwing an `InterruptedException` and a `BluetoothStateException`. The interruption occurs when the lock that we'll set later fails and the Bluetooth state is unable to perform a specific search. We are performing our search in a synchronized block because we do not want to perform any other Bluetooth actions. To start the search, we run the `LocalDevice.getLocalDevice().getDiscoveryAgent().startInquiry(DiscoveryAgent.GIAC, this);` method. The `LocalDevice.getLocalDevice()` returns the local Bluetooth device on the Raspberry Pi. As this Bluetooth device is capable of discovery, we can request the discovery agent with `getDiscoveryAgent()`. We want to immediately start with the search, so we call the `startInquiry` method. There are multiple types of inquiry we can perform. This library only supports the **General Inquiry Access Code** (**GIAC**) method. This method performs a search that the Bluetooth specification allows to run continuously. The other constant is **Limited Inquiry Access Code** (**LIAC**) and may only be used when explicitly asked for by the user, and can only run for a short time. Only GIAC is supported, so we use this constant. We need to register a listener that is being used to call methods on. We are using this class as the listener so we reference it with `this`. The method returns a Boolean when discovery is started, so we check this with `if (started) {`. If this is `true`, we acquire a lock on the declared field for the lock with `searchLock.wait();`. The code will now wait at this point until the `searchLock` is `notified();`. When this is done, the code will continue to run and call the `notifyListeners();` method to notify the listeners the discovery is done. We have registered this class for listening to any events being fired from the discovery thread. Let's look at them.

We first come across `public void deviceDiscovered(RemoteDevice btDevice, DeviceClass btDeviceClass)` which is being called when a device is found. We will only be using the `RemoteDevice btDevice` variable as we only want to know the devices. We are currently not interested in the `device` class. When a device is found, we add it to the list with `discoveredDevices.add(btDevice);`. We want to do as little as possible as we do not want to block the discovery thread, which can cause the Bluetooth stack to lock up, so we return immediately after adding the device. We are not using the `servicesDiscovered` method as we are not interested in the running services when we only want to detect a device's presence. Also, the `serviceSearchCompleted` is not used as we are not discovering services. This method is called when a services discovery on a device has finished. We do want to use the `public void inquiryCompleted(int I)` method as this informs us the discovery has ended. The I parameter is to inform us how the discovery ended.

Up until now, I have only received the `INQUIRY_COMPLETED` constant, which is 0 (0x00), which means a successful inquiry. In the `inquiryCompleted` method, we release the lock with `searchLock.notify();` in a synchronized block so the `startSearch()` method will continue executing after the `searchLock.wait();` method and will call `notifyListeners();` to update the `listening` class. What happens in the `listening` class in our example was explained previously.

We are almost there! We have our display running, our LDR detecting light intensity, and we are able to switch using a mechanical relay. We also already have automated the relay based on the ambient lighting in the surroundings. We are closing in on the final step, where we want to be able to only switch the relay when our phone is detected.

Putting it all together, our first automation project

It is time to put it all together. As we explained, we want to switch based on presence detection and only when the ambient light reaches a certain level. This is a code-only section. We will be walking through the code and will code some lines ourselves.

If you haven't already, write down or copy the device's Bluetooth address by running the `runBluetoothDetectionExample();` one more time as we will be using this address to get a positive detection. When we have the address, we will be writing a couple of lines of code that run all the components we have looked at as a single application. Let's close all open tabs and only open the `Main` class. Comment the `runBluetoothDetectionExample();` method and scroll to the bottom of the class, where we will be adding the following code:

```
/**
 * Runs the application.
 * @throws LcdSetupException When the LCD is not able
    to be initialized.
 */
public static void runApp() throws LcdSetupException{
    final App app = new App();
    Runtime.getRuntime().addShutdownHook(new Thread()
{
        @Override
        public void run() {
            app.destroy();
            GpioFactory.getInstance().shutdown();
        }
    });
```

```
            app.init();
            app.run();

        }
```

This little piece of code runs the application with all the components. This code is normal class initialization and calling some methods, but we are adding some special code that is executed when the JVM shuts down. We register a shutdown hook in the JVM with `Runtime.getRuntime().addShutdownHook(Thread thread);`. We must be sure that this shutdown hook contains very quick and short methods for it to be able to run very quickly.

The shutdown hook is only called during normal JVM shutdown.

In our shutdown hook, we destroy any initializations in the `App` class that we have running and stop all monitoring and other threads running within the GPIO libraries of `Pi4J`. When we have put the previous code in our `Main` class, NetBeans is unable to find the `App` class. This is logical as it is not in the same package as our `Main` class. The following key combination comes in handy when we need to find our imports. Press *Shift* + *command* + *I* on a Mac and *Shift* + *Ctrl* + *I* on other keyboards. This will find the necessary imports automatically, including the `App.java` class, and cleans up unused imports. Now we put the `runApp();` method inside the `main(String[] args)` with the following code:

```
try {
    runApp();
} catch (LcdSetupException ex) {
    System.err.println("Unable to run the app: " +
    ex.getMessage());
}
```

This will run the `runApp()` method with all the components working. When the LCD cannot be initialized, we just exit the app as we want to see what is being registered on the LCD display. But before we start to run the app, let's open the `App.java` class and see what is happening there.

When we have opened the `App.java` class by following it or opening it from the `chapter2.app` package, we will see the flow followed to detect everything we need to be able to switch the relay. When we take a look at the top of the class, we come across a couple of well-known classes we have been looking at in the previous sections, which are `LcdHandler`, `LdrReader`, `RelaySwitch`, and `BluetoothDevices`. Below these, we start three thread pools, one to execute two different threads with `ScheduledExecutorService servicesScheduler = Executors.newScheduledThreadPool(2);`. There are also two single thread executors that will be running code continuously created on the `displayWriter` fields, and `deviceSearch` which will respectively be the thread to write to the LCD and running the Bluetooth discovery. When we scroll down further, we see a couple of fields responsible for holding some state information for the decision making in the code.

The Boolean `running` field is used to determine if we are running or not, which will be used in the Bluetooth detection loop. The `isDark` field is also a Boolean to keep track of the ambient lighting detected, `false` for light and `true` for dark. The `deviceFound` field is used to check if there is a device found and we want to use to switch the relay. The last field in this list is the `allowedDevice` field, which is the one where we will be putting the address found in the discovery. When this device is found, the `deviceFound` field will be set to `true` and when `isDark` is `true`, the relay will be switched to closed.

The constructor in the class only contains initializations we have been using earlier. In the `init` method, we do the same as in an earlier section with initialization of the listeners in the `Bluetooth` class. After init, we call the `run()` method in the `App` class from which we call other methods to start the components. We start with the contents of the `setLdrScheduler()` method:

```
servicesScheduler.scheduleAtFixedRate(() -> {
    if(ldrReader.singleDarkDetect(false) == true){
        isDark = true;
        writeLcd(0, 12, "Dark ");
    } else {
        isDark = false;
        writeLcd( 0, 12, "Light");
        this.relaySwitch.openRelay();
    }
}, 0, 10, TimeUnit.SECONDS);
```

This method contains a Runnable that is scheduled at a fixed rate, It immediately does a check if the ambient lighting is identified as dark with `if(ldrReader.singleDarkDetect(false) == true){`. It does this every ten seconds after the method finishes. When it is dark, it sets the `isDark` field to `true`. This field is used in another method we will be looking at later. Next to setting this field, the text `Dark` is written to the LCD. When it is light,`isDark` filed is set to `false`, we write `Light` to the LCD, and immediately open the relay so it breaks the circuit at the other side. With the LDR scheduler running, let's take a look at the `setTimeUpdater()` method:

```
final SimpleDateFormat formatter = new
SimpleDateFormat("HH:mm");
servicesScheduler.scheduleAtFixedRate(() -> {
    writeLcd(0, 0, formatter.format(new Date()));
}, 0, 60, TimeUnit.SECONDS);
```

This method updates the time on the LCD. When submitted to `servicesScheduler` it immediately runs the submitted `Runnable lambda` method, which shows the time on the display, and then does this every 60 seconds. Before the code block is submitted to the scheduler, we first create `formatter`. This must be the last thing as this is required for every method within a `lambda`. We won't be covering the next method, `writeLcd`, as this has been covered in a previous section. The only difference now is we do not check if the field is `null` as this is not possible in this setup. Let's look at the `setBluetoothDetectionScheduler()` method, which runs the Bluetooth detection method, checks the fields for statuses, and acts on them. Again, we submit a `Runnable lambda` code block to a thread pool. This current thread pool is not a scheduler as we would like to check continuously for the presence of the Bluetooth enabled device. This is the reason why it is in a while loop that checks if the `class` field `running` is set to `true`. When we look at the if statement, we see the following code:

```
this.blDevices.startSearch();
for(RemoteDevice device:this.blDevices.getFoundDevices()){
    if(allowedDevice.equals(device.getBluetoothAddress())
    && isDark){
        writeLcd(1, 0, "Detected        ");
        this.relaySwitch.closeRelay();
        deviceFound = true;
    }
}
```

As covered earlier, we know the device search is a blocking method resulting in code halting before the method returns. This makes sure the code is not looping at an extremely high rate because of the while loop. While looking further into this code block, we see the methods appearing we have covered earlier. We check if the device address in the `allowedDevice` field equals one of the addresses found by the discovery service with `if(allowedDevice.equals(device.getBluetoothAddress()) && isDark){`. We also check if the `isDark` field is set to `true`. If these both are the case, we write to the LCD at character position 0 on the second line with `Detected`, we close the relay, and set a local `deviceFound` variable to `true`. The application has found the device. We have done the first half of the detection. The second part of this code is as follows:

```
if(deviceFound == false){
    writeLcd(1, 0, "No device       ");
    this.relaySwitch.openRelay();
}
deviceFound = false;
this.blDevices.clearFoundDevicesList();
```

By default, the `deviceFound` field is set to `false`. So, when a device is not found it stays `false` and will allow us to write `No device` to the display and open the relay. Because we do not only want to open the relay when it's light, we also want to do this when the Bluetooth device is not detected. After we have checked if the device is found or not we set the `deviceFound` value back to `false` as we are done checking. Just in case the device has been found in the current iteration but won't be in the next iteration, we still need to change the relay and text on the display. We also clear the devices list with `this.blDevices.clearFoundDevicesList();` because otherwise, like in the previous case, the device is still found even while it's not. We now have covered all the code for detection and can act on it. Our final method is the `destroy()` method, which unsets all the set fields and stops all thread pools. The `destroy` method also does the same as described in the `Bluetooth devices` class but with some additions. Because we initialize thread pools, it is nice to shut them down with `shutdown();`. We could also do `shutdownNow();` but this will not wait for a thread to end and could possibly stop a method while it is running.

It's time to press **Run** in the editor. The code will compile and when it's running on the Raspberry Pi, try to enable and disable Bluetooth on your phone while it is light or dark near the LDR. You will notice the relay will act according to the Bluetooth and the ambient light status. Awesome.

I do need to make one point. Sometimes it is possible your device is not detected in one of the iterations. This results in the relay switch closing, then opening, and then closing again. This is quite normal. Try to change the code so that instead of immediately opening the relay after the first non-detection of the device into multiple non detections. Three iterations should be fine, Good luck!

Summary

In this chapter, we have interfaced with a 16x2 character display directly connected to the Raspberry Pi and covered the code used to change the content. We have covered how we are able to read analog data with digital pins by implementing an RC circuit. By attaching a relay with a transistor we have seen how we can switch a 5 V device with a 3 V signal pin that in turn is able to turn on a high-power device. And last but not least, we have taken the first steps with the Raspberry Pi integrated Bluetooth chip to detect nearby Bluetooth devices. This has all led us to our very first home automation project, allowing us to detect presence based on ambient lighting to switch a relay.

As you may have noticed, we looked quite deeply into each step of the code. This is meant as the introduction into the foundations of the `Pi4J` libraries and how these are used. We have not covered every part yet, only the basics that will be useful for forthcoming chapters.

In `Chapter 3`, *A Social and Personal Digital Photo Frame*, you will build a multimedia project which is a social and personal digital photo frame using easy to use software and hardware components with Raspberry Pi.

3
A Social and Personal Digital Photo Frame

The Raspberry Pi is a great single-board computer for working with multimedia projects, that is, the integration of graphics, audio, video, and so forth. This chapter presents how to make a social and personal digital photo frame using the Raspberry Pi 3 with a few hardware and software components. It can be mounted on a desktop or attached to a wall, like a traditional photo frame.

The social media platform we'll be going to use is Flickr, which provides a Flickr API to deal with a set of Flickr web services. The photo frame is capable of fetching all the images stored in a specified Flickr album and displaying them as a full-screen slideshow on a video display at regular intervals. Also, the photo frame can periodically sync its local image library with the pointed Flickr album.

In this chapter, we will cover the following topics:

- Connect and configure the Waveshare HDMI display with the Raspberry Pi
- Learn how to create a Flickr app and obtain an API key and secret
- Learn how to create a Flickr album
- Learn how to work with the Flickr API's methods
- Write a Java program to make REST requests and process REST responses
- Install and configure feh on the Raspberry Pi 3 to run a slideshow
- Learn how to work with `crontab` and `rc.local` to automate the application

We are now ready to create our personal digital photo frame that displays a Flickr photo album as a slideshow.

Bill of materials

First, we should prepare the following things to build the project:

- Raspberry Pi 3, Model B: `https://www.modmypi.com/raspberry-pi/accessories/power-supplies/raspberry-pi-official-universal-power-supply-5.1v-2.5a-white/`
- Raspberry Pi Official Universal Power Supply 5.1V, 2.5 A: `https://www.modmypi.com/raspberry-pi/accessories/power-supplies/raspberry-pi-official-universal-power-supply-5.1v-2.5awhite/?search=power%20supply&page=1`
- Waveshare 10.1 inch HDMI LCD display (H) (with case), 1024 x 600: `http://www.waveshare.com/10.1inch-hdmi-lcd-with-case.htm`
- Official Raspberry Pi HDMI-to-HDMI Cable (1 m Black): `https://www.modmypi.com/raspberry-pi/accessories/video-cables/official-raspberry-pi-hdmi-to-hdmi-cable-1m-black`
- Small Phillips head screw driver (precision screwdriver set of six pieces): `https://www.adafruit.com/product/424`

Waveshare HDMI display

The Waveshare 10.1 inch HDMI LCD display is an ideal solution for building your digital photo frame. The display has the following features:

- HDMI, VGA, and AV (CVBS) multi-video interfaces
- Video source can be selected from the built-in OSD menu using control buttons
- Screen resolution is 1024 x 600, which is **Wide Super VGA** (**WSVGA**)

The most expensive hardware you need for this project is the Waveshare 10.1 inch HDMI display. However, you can use any display that has a HDMI interface to connect with your Raspberry Pi.

The Waveshare display is pre-assembled in acrylic and further assembled on an acrylic mounting sheet with a screen controller board. The screen controller board is connected to the screen using a ribbon cable. It also offers a 12V, 2 A EU power supply, mounting bolts, and screen stand:

Figure 3-1: Waveshare display (front view). Image credits: courtesy of ModMyPi (https://www.modmypi.com). Similar: Waveshare (http://www.waveshare.com/)

Assembling with Raspberry Pi

Assembling the Waveshare display to the Raspberry Pi is quite easy, with a few simple steps.

The Waveshare kit includes the following components, shown in the Figure 3-2:

- Screen (pre-assembled in acrylic)
- Acrylic mounting sheet
- Screen controller board
- Ribbon cable
- HDMI cable

- USB-to-microUSB cable
- Screen stand
- Mounting bolts
- Power supply (12V 2 A)

Figure 3-2: Components of Waveshare display. Image credits: courtesy of ModMyPi (`https://www.modmypi.com`)

The complete assembly guide of the Waveshare display can be found at `https://www.modmypi.com/blog/waveshare-101-screen-assembly-guide`. Please follow the given instructions correctly to assemble the display with other hardware components such as screen driver board and Raspberry Pi.

Selecting video source

The screen has a built-in OSD menu for video source selection, power management, and brightness and contrast adjustment. The OSD menu can be controlled with four push buttons attached to the screen controller board (Figure 3-3):

Figure 3-3: OSD menu control buttons. Image credits: courtesy of ModMyPi (`https://www.modmypi.com`)

1. Press the **Source** button to see the video source selection menu on the screen. Press the **Left** and **Right** buttons, followed by the **Menu** button, to select **HDMI** as the video source (Figure 3-4):

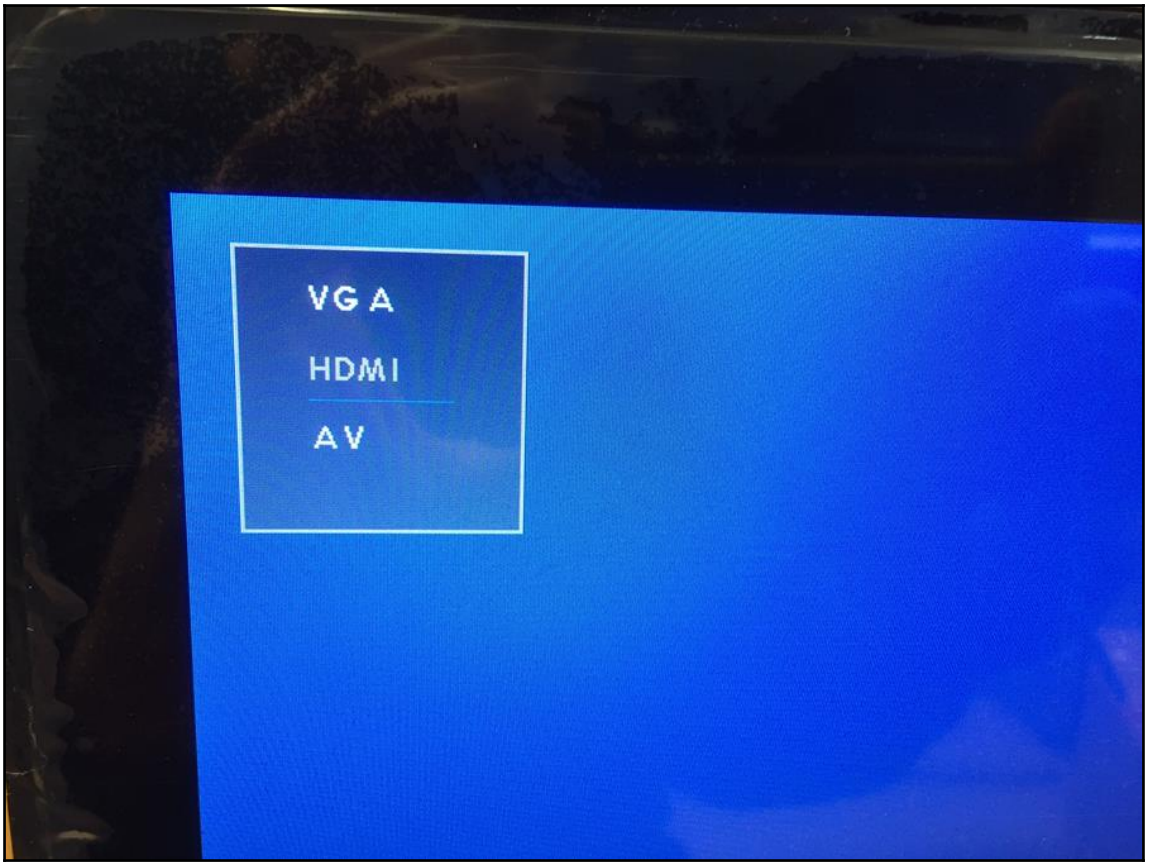


Figure 3-4: HDMI option. Image credits: courtesy of ModMyPi (`https://www.modmypi.com`)

2. Apply power to your Raspberry Pi. The Pi boots and you can see the Raspbian desktop on the screen (Figure 3-5):

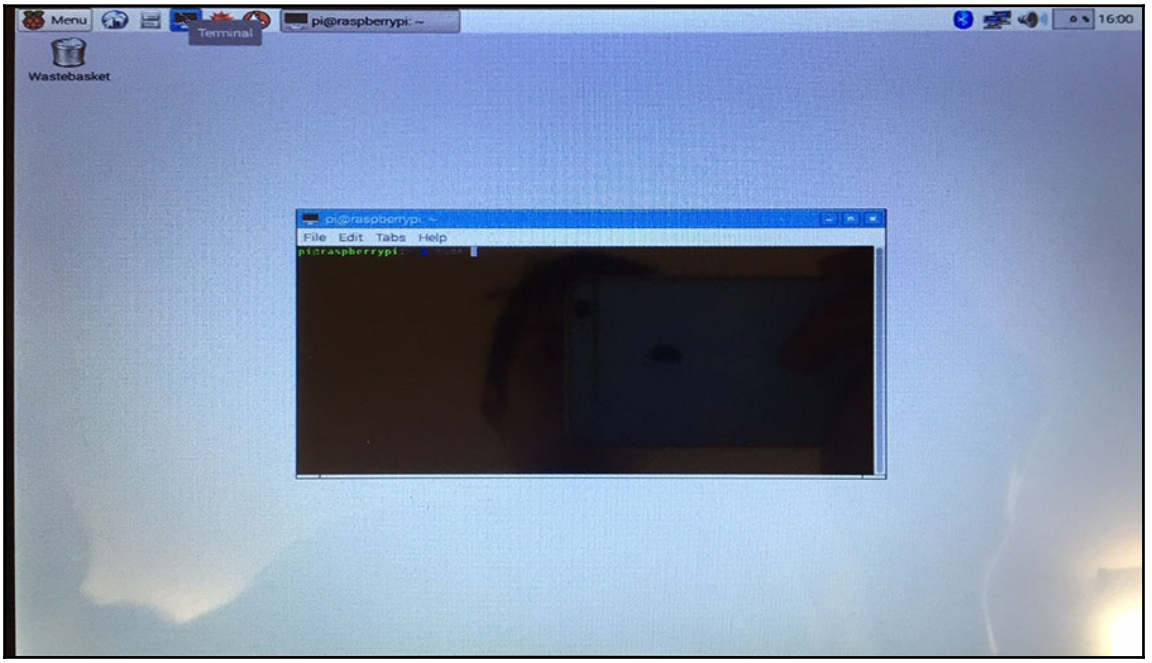


Figure 3-5: Raspbian desktop on Waveshare display. Image credits: courtesy of ModMyPi (`https://www.modmypi.com`)

Now you're ready to move to the next configuration step, which is correcting the display resolution.

Correcting display resolution

The first time you boot the Raspberry Pi, the Waveshare display will start with an incorrect resolution.

The following steps will help you to correct this using a few configuration settings:

1. Open the Raspberry Pi **Terminal** by clicking the **Terminal** icon in the **Application Launch Bar**.
2. Open the boot config file (`config.txt`) with the Nano text editor using the following command (Figure 3-6):

```
sudo nano /boot/config.txt
```

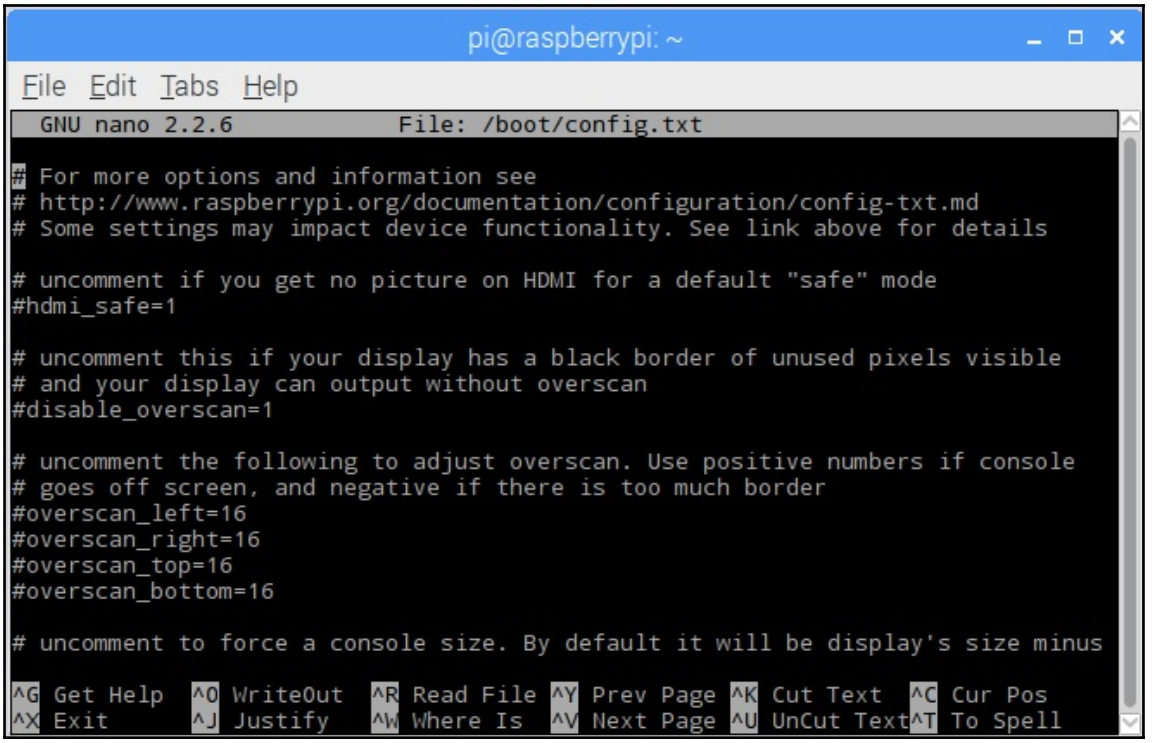


Figure 3-6: config.txt file opened with the Nano text editor

3. Then, add the following settings at the end of the file (Figure 3-7):

```
max_usb_current=1
hdmi_group=2
hdmi_mode=1
hdmi_mode=87
hdmi_cvt 1024 600 60 6 0 0 0
```

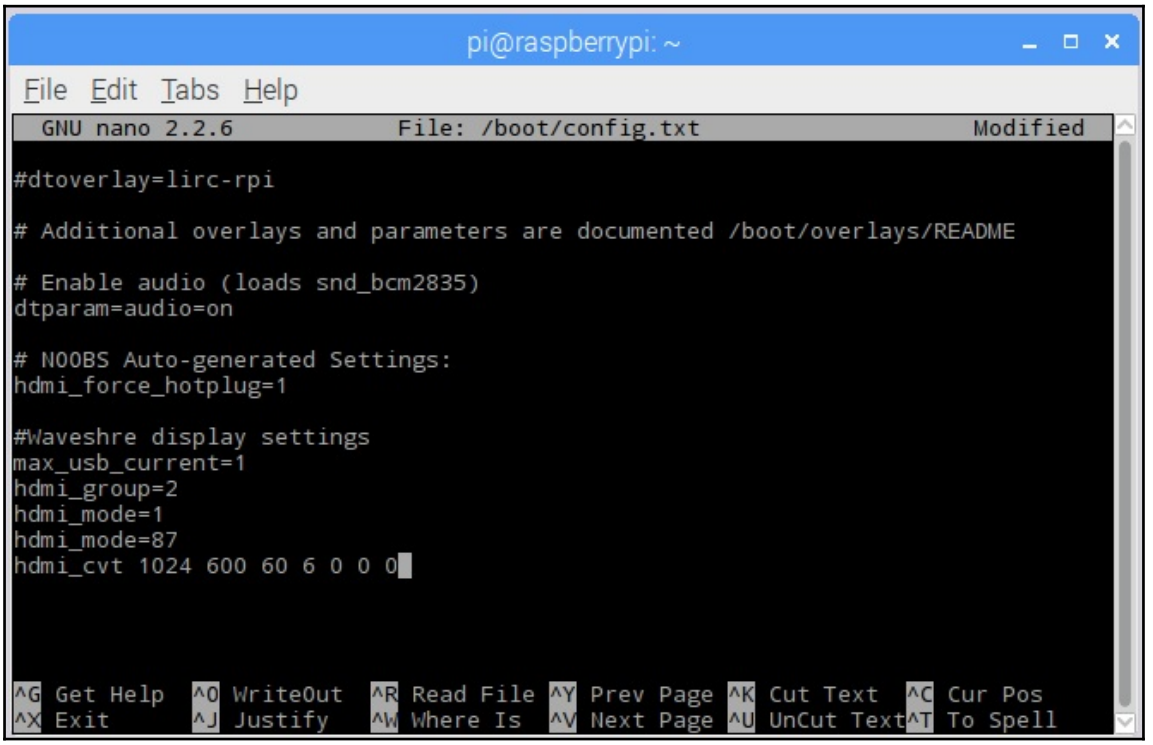


Figure 3-7: Display settings for Waveshare display

4. Save the file by pressing *Ctrl+O* on your keyboard, followed by the *Enter* key.
5. Then, exit the Nano editor by pressing *Ctrl + X*.
6. Finally, reboot the Raspberry Pi by using the following command:

```
sudo reboot
```

Alternatively, you can reboot the Pi by clicking on **Menu | Shutdown**, followed by clicking on the **Reboot** button from the dialog box.

Now you can see that the screen was restored with the correct resolution, as shown in the Figure 3-8:

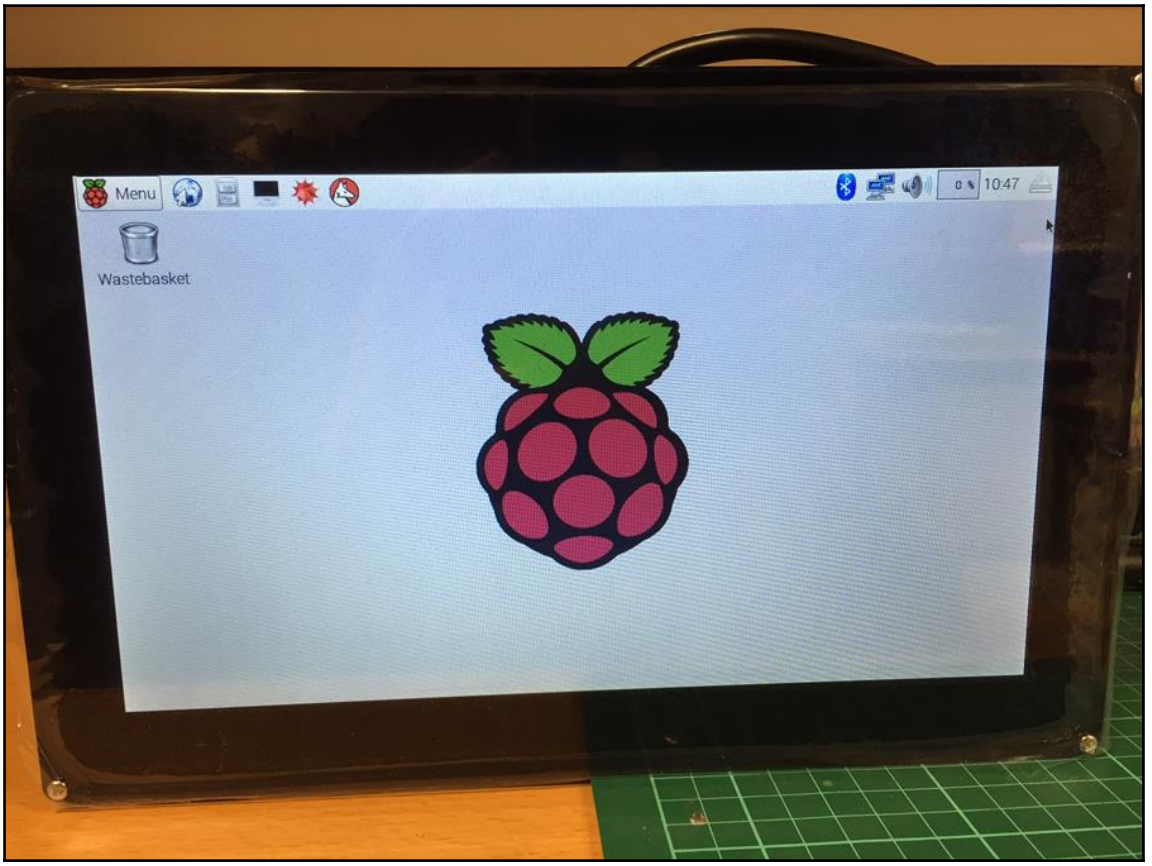


Figure 3-8: Display started with correct resolution. Image credits: courtesy of ModMyPi (`https://www.modmypi.com`)

Great! You have successfully corrected the Waveshare display resolution to work with the photo frame.

Mounting on desktop

The package includes two screen mounts. Carefully place the screen on the screen mounts, as shown in the Figure 3-9:

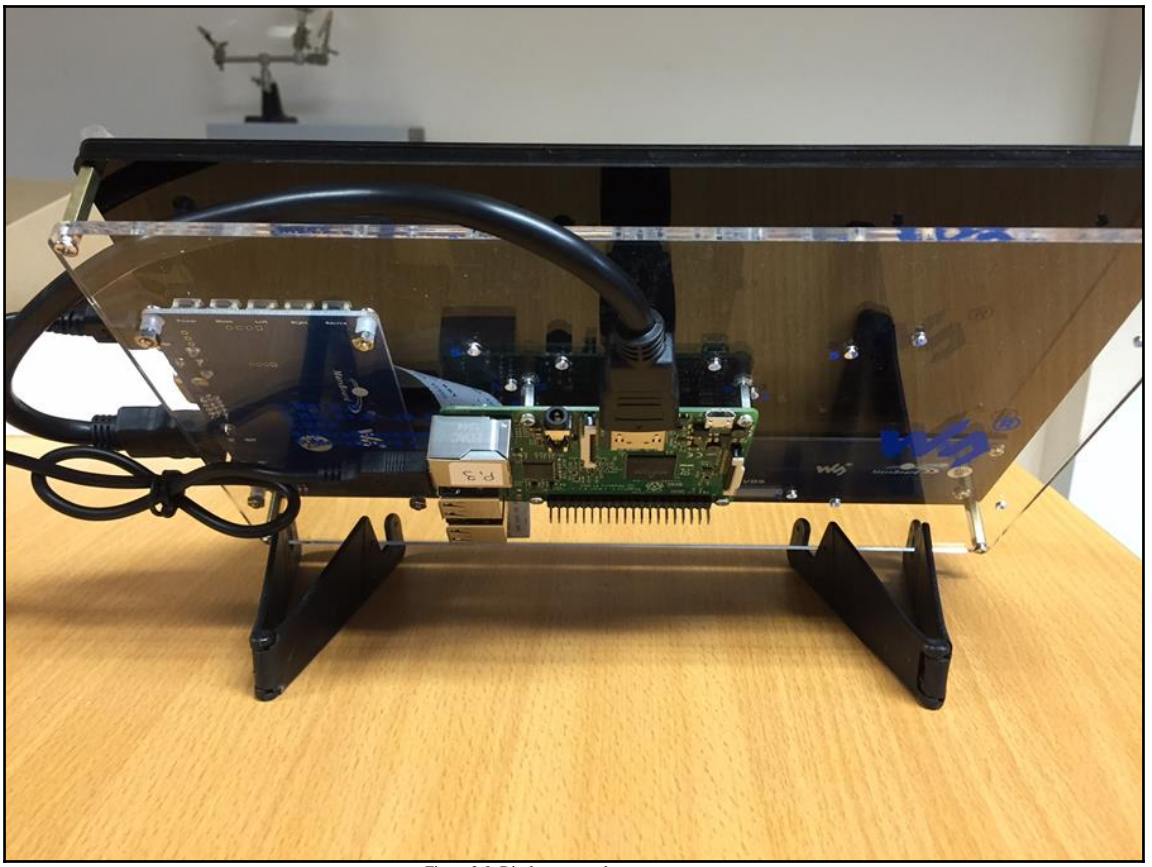


Figure 3-9: Display mounted on screen mounts

Instead of mounting the display on a desk or table, you can attach it to the wall by making a 3D printed or wooden frame. It's up to you!

The hardware setup for the photo frame is now completed. Let's move on to the software that is needed to write, install, and configure on Raspberry Pi.

Connecting with Flickr

Flickr is a photo sharing and hosting service that enables you to store and share your photos and videos with your family and friends, the public, or keep them completely private. It allows you to organize your photos and videos into albums.

Obtaining a Flickr API key

You have to obtain an API key from Flickr by signing in with your Yahoo account before accessing any Flickr API method. If you don't have one, sign up with Flickr to create a new Yahoo account. The following steps will guide you through how to obtain an API key:

1. Using your web browser, visit **The App Garden** at `https://www.flickr.com/services/`.
2. Click on the **Create an App** link (Figure 3-10):

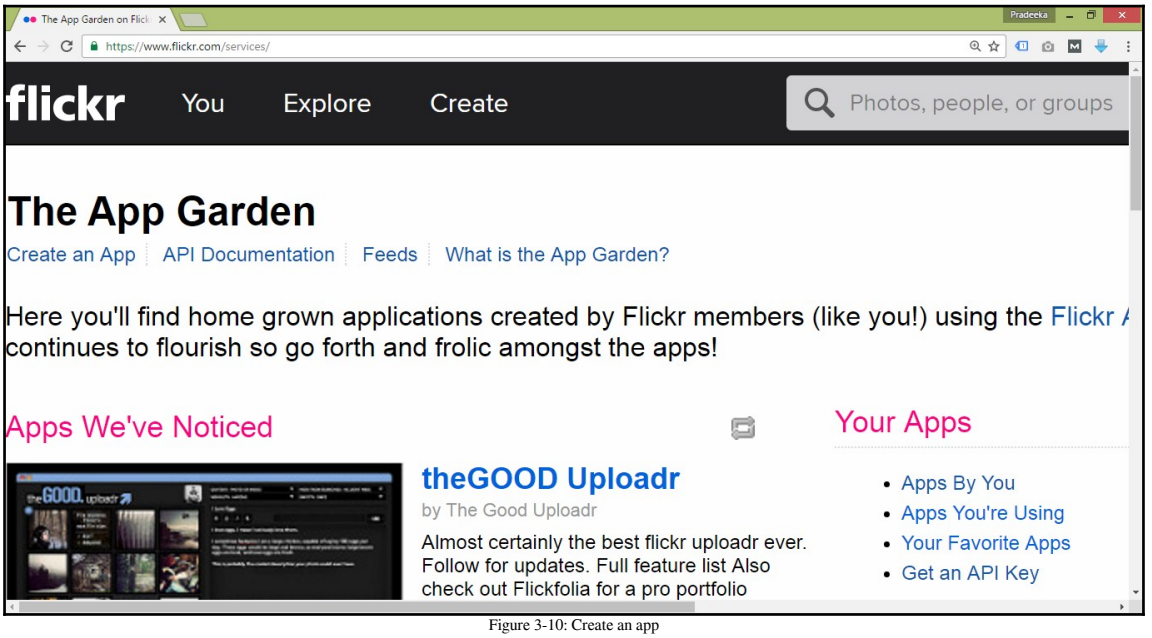


Figure 3-10: Create an app

3. Click on the **Request an API Key** link (Figure 3-11):

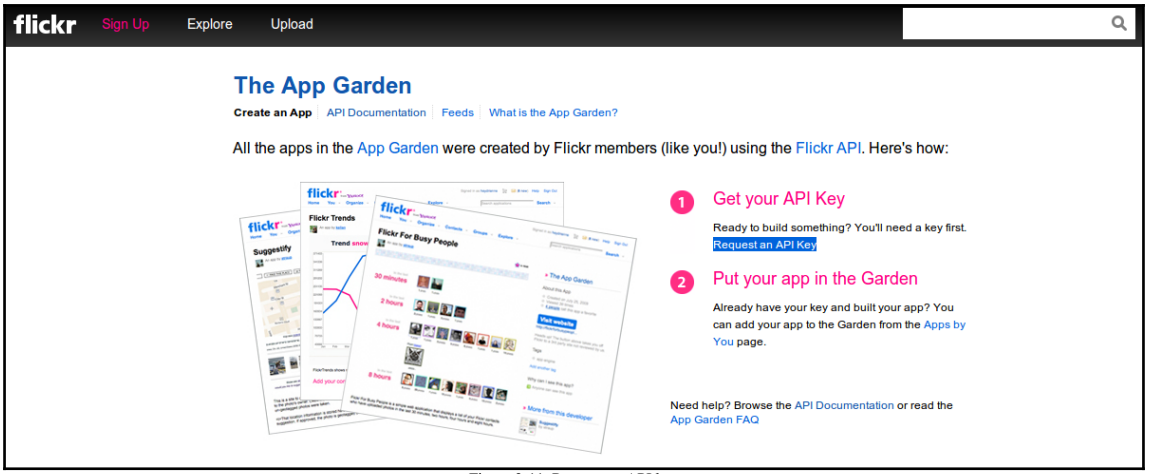


Figure 3-11: Request an API key

4. Click on the **APPLY FOR A NON-COMMERCIAL KEY** button to choose the non-commercial key option, because we're going to use the app only for personal use (Figure 3-12):

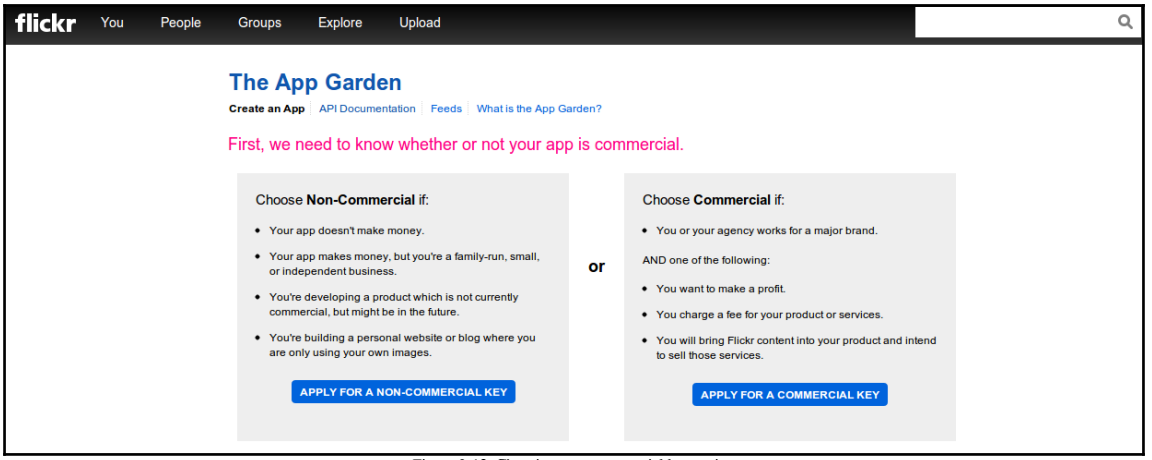


Figure 3-12: Choosing non-commercial key option

5. The **Tell us about your app** page will appear. Fill in the following fields, accept the terms and conditions (two checkboxes), and click the **Submit** button to create the app (Figure 3-13):
    - **What's the name of your app?**: Raspberry Pi Digital Photo Frame
    - **What are you building?**: Display photos on a Raspberry Pi connected to a HDMI screen as a slideshow

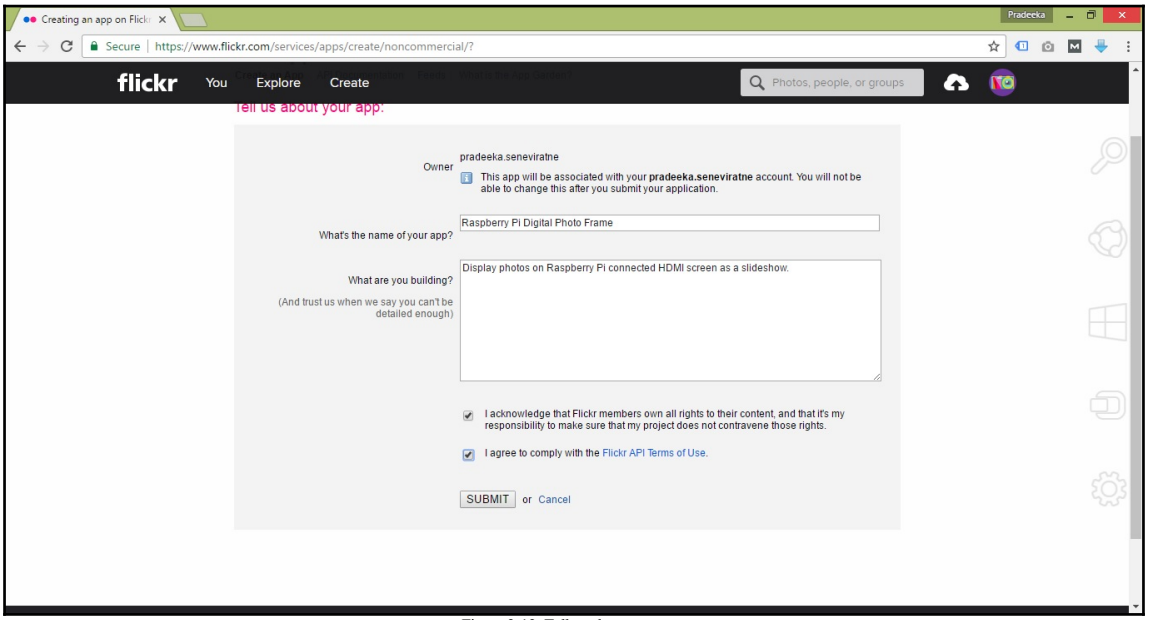


Figure 3-13: Tell us about your app

6. You will get an API key and secret similar to those shown in the Figure 3-14. Furthermore, you can edit the app by clicking on the app title (**Raspberry Pi Digital Photo Frame**):

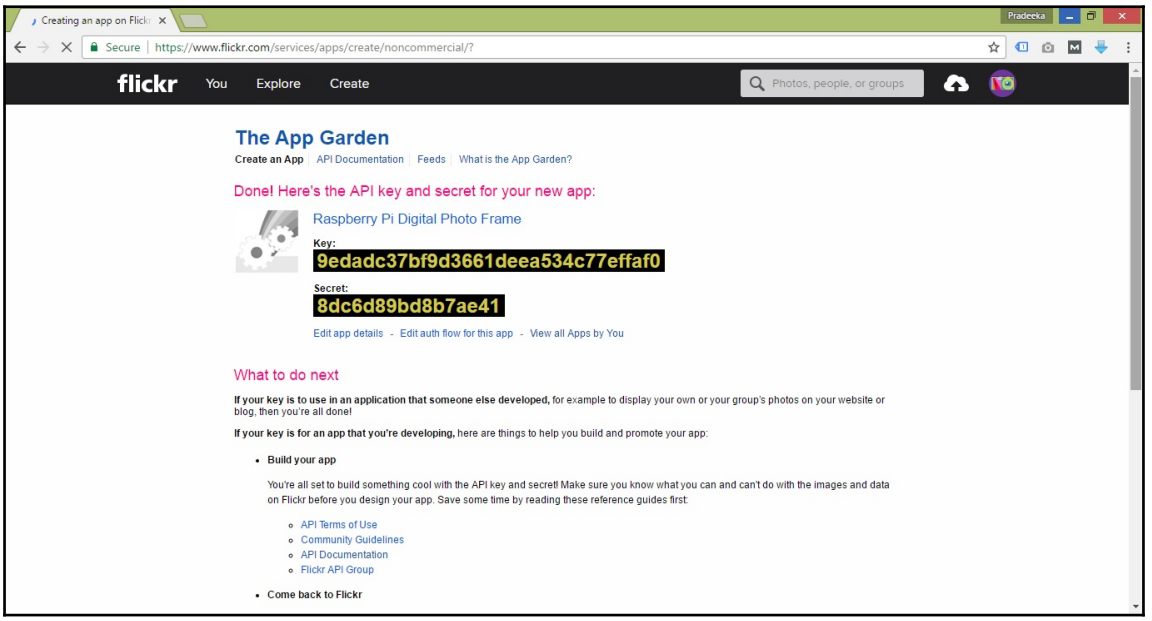


Figure 3-14: Key and secret for app

- **Key**: `112c7bffb76bc5f5660a75fc0212478e`
- **Secret**: `c345efee374cbe13`

The key and secret are very important when you are working with the Flickr API methods.

Creating an album

To create an album, you should have at least one photo in your Flickr photo stream. The following steps explain how to create a new album:

1. In the Flickr main menu, mouse over **You** and click **Organize** from the drop-down menu.
2. Click on the **Albums & Collections** tab.

3. Click **Create a new album**, at the top of the page.
4. Type a title and description in the left column.
5. Drag content into the album area from the bottom. This area shows the contents in your photo stream (Figure 3-15):

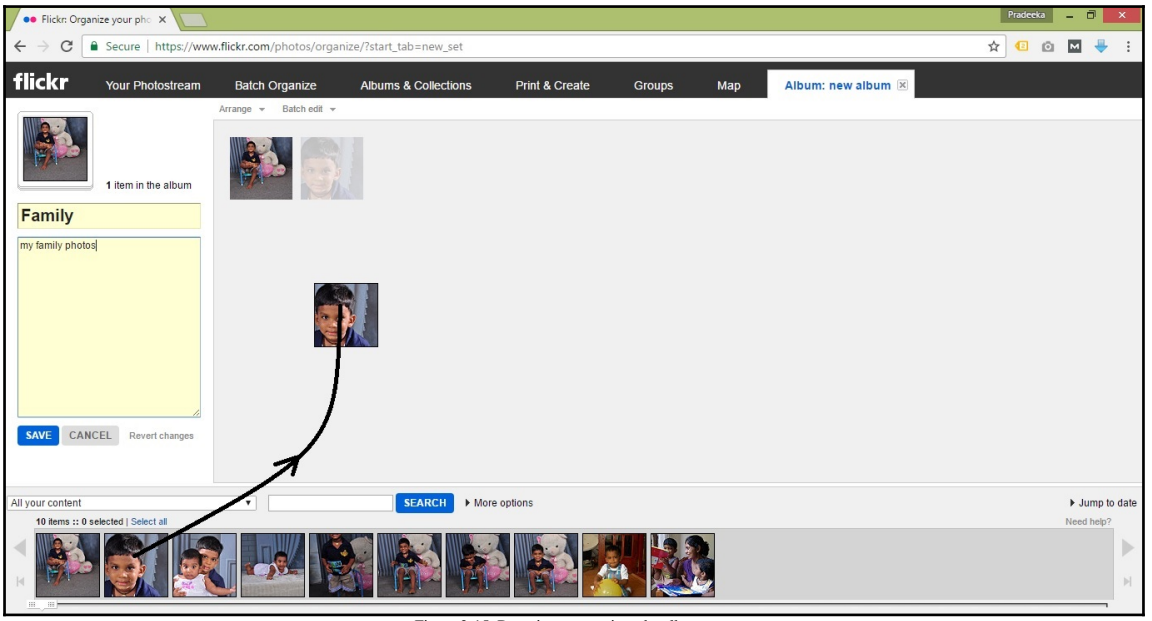


Figure 3-15: Dragging content into the album area

6. Click the **Save** button.

Finding Flickr photoset_id

The Flickr `photoset_id` is a unique ID associated with a Flickr photo album and can be used to fetch information from a photo set (also known as an album). First, make sure that you have at least one photo album in your Flickr account.

The terms `photoset` and `album` are used interchangeably.

In the Flickr main menu bar, click **You** | **Albums**. Flickr will show all the albums that you have created with your account. Click on one of your favorite albums that you'd like to use with the photo frame. In the address bar of your web browser, at the end of the URL, you can find the `photoset_id` associated with your album `https://www.flickr.com/photos/128978031@N04/albums/72157659233917324`.

The `photoset_id` appended to the previous URL is `72157659233917324` (Figure 3-16):

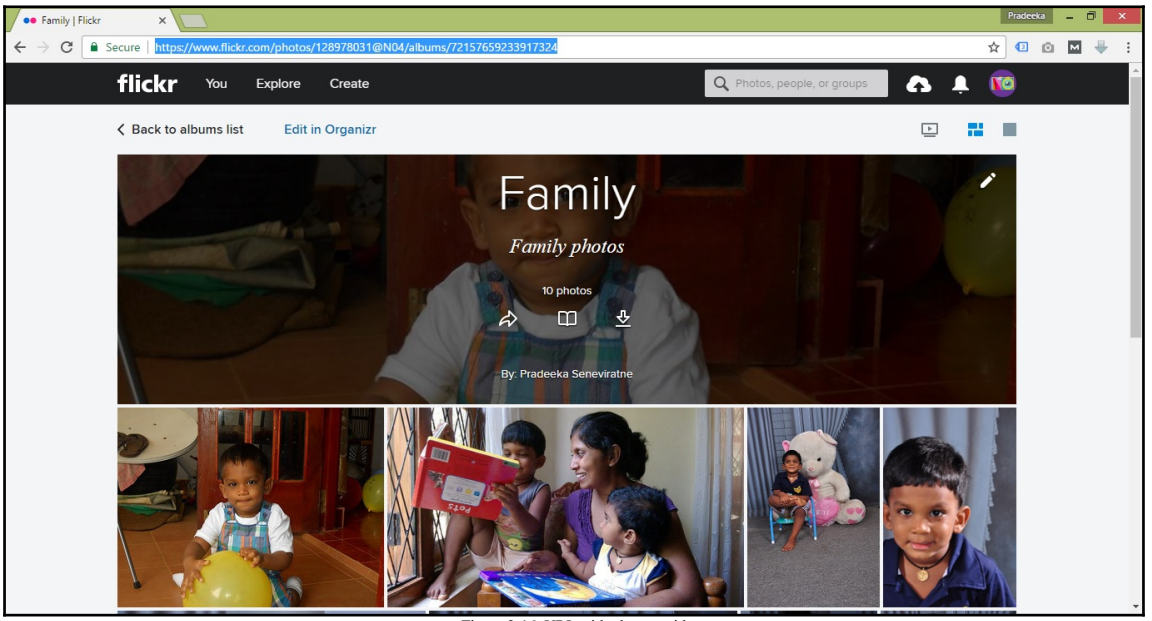


Figure 3-16: URL with photoset_id

REST request format

The Flickr REST endpoint URL is: `https://api.flickr.com/services/rest/`. You can find all the Flickr API methods at `https://www.flickr.com/services/api/`.

Invoking flickr.test.echo

The `flickr.test.echo` API method can be used to test your Flickr app and it echoes all parameters back in the response. This method doesn't require any authentication, and the only required parameter is the `api_key` that is associated with your Flickr app. We've passed additional parameters (`name=foo` and `value=bar`) with the URL:

```
https://api.flickr.com/services/rest/?method=flickr.test.echo&api_key=9edad
c37bf9d3661deea534c77effaf0&foo=bar
```

You should see something similar to the following:

```
<rsp stat="ok">
<method>flickr.test.echo</method><api_key>9edadc37bf9d3661deea534c77effaf0<
/api_key>
<foo>bar</foo>
</rsp>
```

The response XML payload contains all the parameters echoed by the server as elements:

```
<method>flickr.test.echo</method><api_key>9edadc37bf9d3661deea534c77effaf0<
/api_key>
<foo>bar</foo>
```

Invoking flickr.photosets.getPhotos

The `flickr.photosets.getPhotos` API method can be used to get a list of photos in an album. The URL for the REST request can be written as follows:

```
https://api.flickr.com/services/rest/?method=flickr.photosets.getPhotos&api
_key=9edadc37bf9d3661deea534c77effaf0&photoset_id=72157659233917324&user_id
=128978031@N04
```

You should get a REST response XML payload similar to the following, according to the content of your album:

```
<rsp stat="ok">
<photoset id="72157659233917324" primary="32167210005"
owner="128978031@N04" ownername="pradeeka.seneviratne" page="1"
per_page="500" perpage="500" pages="1" total="3" >
<photo id="32167210005" secret="b36c4d132a" server="461" farm="1"
isprimary="0" ispublic="1" isfriend="0" isfamily="0"/>
<photo id="32018670702" secret="8f9e34afed" server="727" farm="1"
isprimary="0" ispublic="1" isfriend="0" isfamily="0"/>
<photo id="31356871303" secret="cc8bb17d64" server="551" farm="1"
isprimary="0" ispublic="1" isfriend="0" isfamily="0"/></photoset>
```

```
</rsp>
```

The `flickr.photoset.getphotos` API method doesn't require authentication and only returns public photos of an album from any Flickr user's account. If you want to get private photos, first you should authenticate your application with Flickr.

Constructing photo source URL

You can construct the source URL for a photo once you know its ID, server ID, farm ID, and secret, as returned by many API methods. The URL takes the following formats:

- `https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg`

- `https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}_[mstzb].jpg`
- `https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)`

Now, let's examine the first photo element of the XML payload returned with the REST response:

```
<photo id="32167210005" secret="b36c4d132a" server="461" farm="1"
 isprimary="0" ispublic="1" isfriend="0" isfamily="0"/>
```

The photo element consists of the `id`, `secret`, `server`, `farm`, `title`, `isprimary`, `ispublic`, `isfriend`, and `isfamily` attributes. The following attribute values can be used to construct the basic image URL:

```
farm-id: 1
 server-id: 461
 photo-id: 32167210005
 secret: b36c4d132a
```

The image URL can be written as follows:

```
https://farm6.staticflickr.com/5698/22623991063_749d617ae0.jpg
```

The constructed image URL can also be written with the size suffix letter `b`, which indicates large, 1024 on the longest side, and can be written as follows:

```
https://farm6.staticflickr.com/5698/22623991063_749d617ae0_b.jpg
```

The complete set of size suffixes are listed in the following table:

Suffix	**Description**
`s`	Small square 75 x 75
`q`	Large square 150 x 150
`t`	Thumbnail, 100 on the longest side
`m`	Small, 240 on longest side
`n`	Small, 320 on the longest side
`-`	Medium, 500 on the longest side
`z`	Medium 640, 640 on the longest side
`c`	Medium 800, 800 on the longest side
`b`	Large, 1024 on the longest side
`h`	Large 1600, 1600 on the longest side
`k`	Large 2048, 2048 on the longest side
`o`	Original image; either a JPG, GIF or PNG, depending on the source format

Writing Java program

Now you're ready to write your Java program for your Raspberry Pi digital photo frame. The following steps will explain how to set up the project with NetBeans:

1. Open your NetBeans IDE, and in the menu bar click **File** | **New Project** (Figure 3-17). The **New Project** wizard will open with Step 1, then choose **Project**. Under the **Categories** tree view, click **Java** and under the **Projects** list view, click **Java Application**. Then, click the **Next** button to move to step 2:

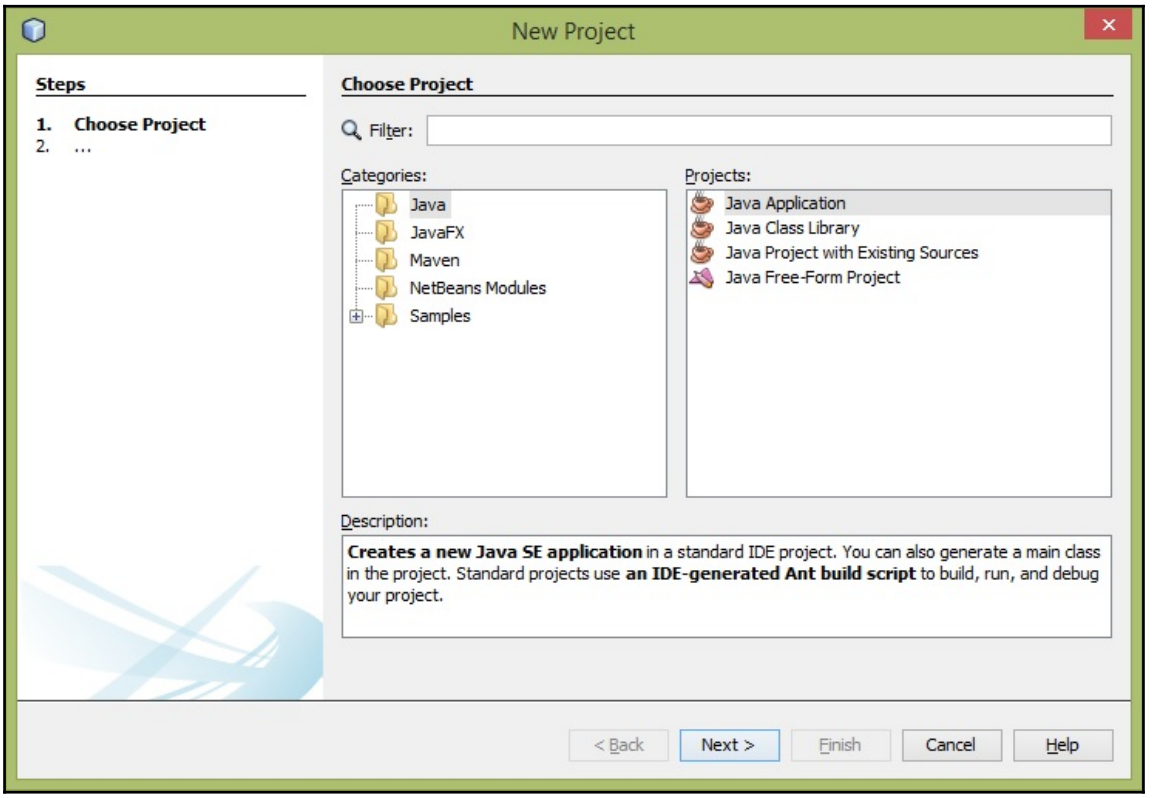


Figure 3-17: Choose project

2. In the **Project Name** text box, type `DigitalPhotoFrame`, and in the **Create Main Class** text box, type `com.packt.B05688.chapter3.DigitalPhotoFrame`. Leave the **Project Location** box as it is to use the default project location in your local computer, or click the **Browse** button to choose a new location. Click the **Finish** button to create the project (Figure 3-18):

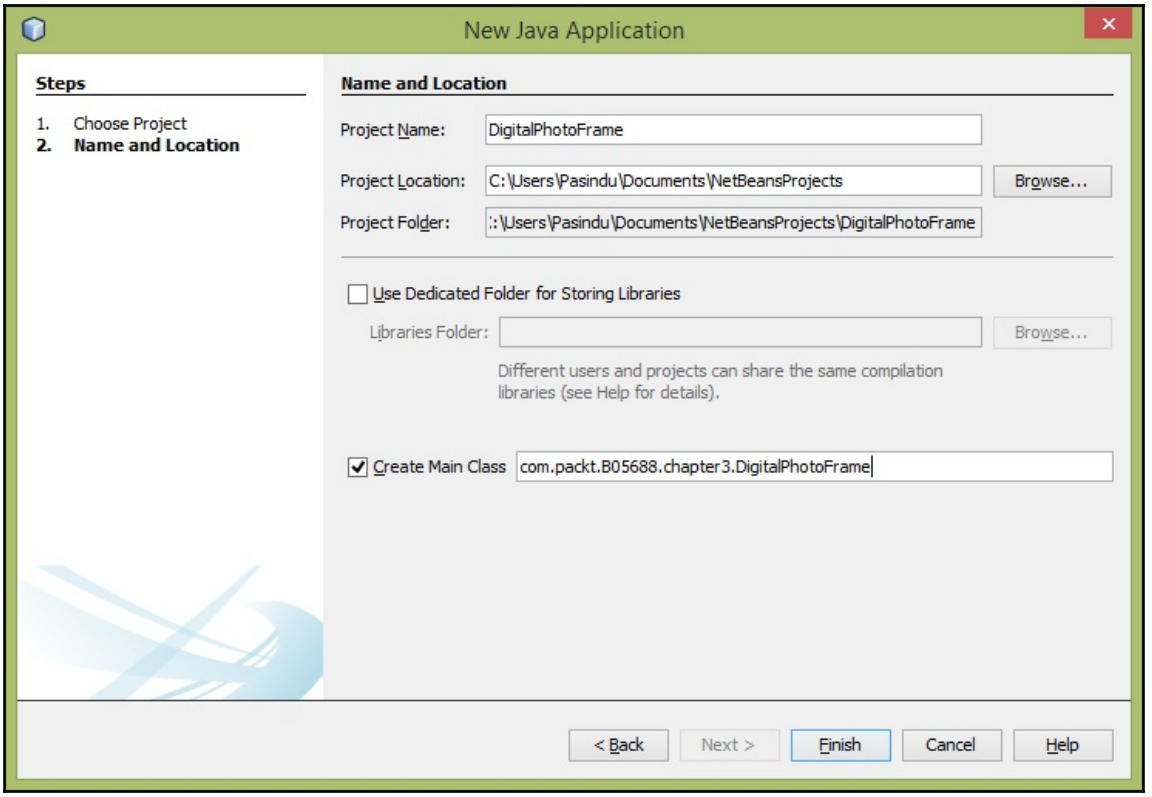


Figure 3-18: Name and Location

3. The NetBeans IDE creates all the necessary packages and files for you. The `DigitalPhotoFrame.java` class file will open with the editor as the main class. Delete all the default code lines in the file.
4. Then, enter each line from Listing 3-1:

```
Listing 3-1: DigitalPhotoFrame.java
package com.packt.B05688.chapter3;
import java.io.*;
import java.net.*;
import javax.xml.parsers.*;
```

```
import org.w3c.dom.*;

public class DigitalPhotoFrame {

public static void main(String[] args) {

try {
URL resourceUrl = new
URL("https://api.flickr.com/services/rest/?method=flickr.photos
ets.getPhotos&api_key=9edadc37bf9d3661deea534c77effaf0&photoset
_id=72157659233917324&user_id=128978031@N04&privacy_filter=1");
HttpURLConnection conn = (HttpURLConnection)
resourceUrl.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {
throw new RuntimeException("Failed : HTTP error code : "
+ conn.getResponseCode());
}

InputStream responseStream = conn.getInputStream();

try {

DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(responseStream);
NodeList nodeList = doc.getElementsByTagName("photo");

for (int x = 0, size = nodeList.getLength(); x < size; x++) {
String s = "https://farm" +
nodeList.item(x).getAttributes().getNamedItem("farm").getNodeVa
lue() + ".staticflickr.com/" +
nodeList.item(x).getAttributes().getNamedItem("server").getNode
Value() + "/" +
nodeList.item(x).getAttributes().getNamedItem("id").getNodeValu
e() + "_" +
nodeList.item(x).getAttributes().getNamedItem("secret").getNode
Value() + "_b.jpg";

System.out.println(s);

URL mediaUrl = new URL(s);
InputStream in = new
BufferedInputStream(mediaUrl.openStream());
OutputStream out = new BufferedOutputStream(new
```

```
FileOutputStream("/home/pi/RASPI3JAVA/DigitalPhotoFrame/media/"
+
nodeList.item(x).getAttributes().getNamedItem("id").getNodeValu
e() + ".jpg"));

for (int i; (i = in.read()) != -1;) {
out.write(i);
}
in.close();
out.close();
}
} catch (Exception e) {
e.printStackTrace();
}

conn.disconnect();

} catch (MalformedURLException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}

}
}
```

Accessing Flickr image URL

Let's examine the Java program step by step to understand the concepts behind the screen. The `DigitalPhotoFrame` project consists of a single file, named `DigitalPhotoFrame.java`.

The `DigitalPhotoFrame.java` file is part of the `com.packt.B05688.chapter3` package, which begins with this:

```
package com.packt.B05688.chapter3;
```

Then, import all the necessary packages to the class:

```
import java.net.*;
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
```

Then, create a URL object from: `https://api.flickr.com/services/rest/?method=flickr.photosets.getPhotos&api_key=9edadc37bf9d3661deea534c77effaf0&photoset_id=72157659233917324&user_id=128978031@N04&privacy_filter=1` using the URL constructor, `public URL(String spec) throws MalformedURLException`:

```
URL resourceUrl = new
URL("https://api.flickr.com/services/rest/?method=flickr.photosets.getPhotos&api_key=9edadc37bf9d3661deea534c77effaf0&photoset_id=72157659233917324&user_id=128978031@N04&privacy_filter=1");
```

URL constructors throw a `MalformedURLException` if the arguments to the constructor refer to a null or unknown protocol. Typically, you want to catch and handle this exception by embedding your URL constructor statements in a `try/catch` pair, like this:

```
catch (MalformedURLException e) {
e.printStackTrace();
}
```

Next, use the `httpURLConnection` class to create an instance to make a single request from the URL object. Then, use the `openConnection` method of the URL class to return a `URLConnection` instance that represents a connection to the remote object referred to by the URL:

```
HttpURLConnection conn = (HttpURLConnection)
resourceUrl.openConnection();
```

Set the method for the URL request to `GET` using `setRequestMethod`, and set the general request property to `Accept`, `application/json` using `setRequestProperty`:

```
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
```

You can handle any runtime exceptions by reading the response code of the request using the `getResponseCode` method, and throw an exception if the response code is not equal to `200`:

```
if (conn.getResponseCode() != 200) {
throw new RuntimeException("Failed : HTTP error code : "
+ conn.getResponseCode());
}
```

Create an object of the InputStream class to handle the response. The `getInputStream` method returns an input stream that reads from the connection that is currently opened:

```
InputStream responseStream = conn.getInputStream();
```

Next, write the XML payload returned by the response stream into a document using the `DocumentBuilderFactory` and `DocumentBuilder` classes:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(responseStream);
```

Then you can build a node list that starts with `photo`, using the `NodeList` class:

```
NodeList nodeList = doc.getElementsByTagName("photo");
```

You can traverse all the nodes in the list using a simple for loop to construct all the image URLs. For debugging purposes, you can print all the constructed URLs on the console using the `println` method:

```
for (int x = 0, size = nodeList.getLength(); x < size; x++) {
String s = "https://farm" +
nodeList.item(x).getAttributes().getNamedItem("farm").getNodeValue() +
".staticflickr.com/" +
nodeList.item(x).getAttributes().getNamedItem("server").getNodeValue() +
"/" + nodeList.item(x).getAttributes().getNamedItem("id").getNodeValue() +
"_" +
nodeList.item(x).getAttributes().getNamedItem("secret").getNodeValue() +
"_b.jpg";
System.out.println(s);
```

Now its time to download all the images from the server. The `FileOutputStream` class constructor can be used to create new files with unique filenames. You can also define the write operation as `append` with a second parameter by using `true`:

```
URL mediaUrl = new URL(s);
InputStream in = new BufferedInputStream(mediaUrl.openStream());
OutputStream out = new BufferedOutputStream(new
FileOutputStream("/home/pi/RASPI3JAVA/DigitalPhotoFrame/media/" +
nodeList.item(x).getAttributes().getNamedItem("id").getNodeValue() +
".jpg"));
```

The `OutputStream` class writes the file to disk using the write method, where the `i` variable holds the bytes to this output stream returned by the `InputStream` variable `in`. The `write` method:

```
for ( int i; (i = in.read()) != -1; ) {
out.write(i);
}
```

Then close the input and output stream objects using the `close` method:

```
in.close();
out.close();
```

Finally, disconnect the `HttpURLConnection` object, `conn`, using the `disconnect` method:

```
conn.disconnect();
```

When you run this application with NetBeans, the program will write all the public images in your Flickr album to the SD card of the Raspberry Pi. The directory is located at `/home/pi/RASPI3JAVA/DigitalPhotoFrame/media/` in your SD card. The output window of the NetBeans IDE prints all the public image URLs, as shown in the Figure 3-19:

Figure 3-19: Output window shows list of image URLs for public photos

You can use the file manager to browse the `/home/pi/RASPI3JAVA/DigitalPhotoFrame/media/` directory and open the saved images using any image processing application installed in the Raspbian OS (Figure 3-20):

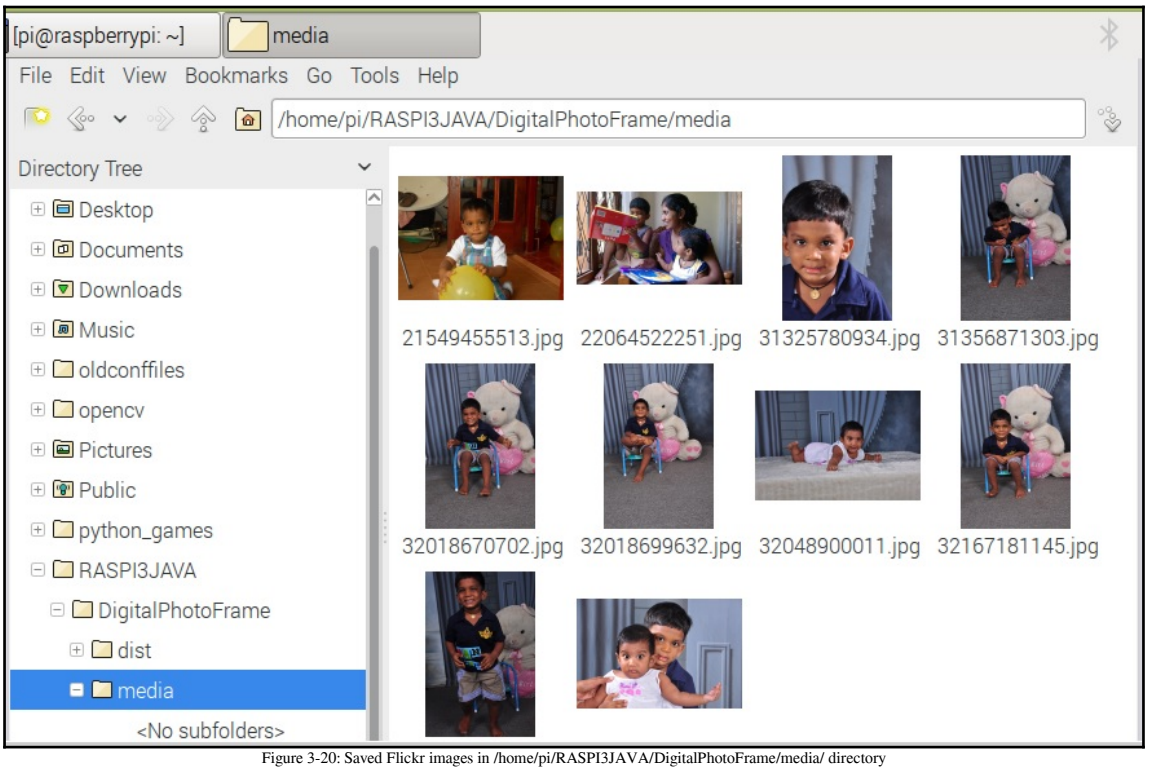


Figure 3-20: Saved Flickr images in /home/pi/RASPI3JAVA/DigitalPhotoFrame/media/ directory

Now we have all the images in the media directory. Let's find an application that can be used to make a slideshow to display images in full screen mode.

Installing feh on Raspberry Pi

feh is a powerful, fast, and lightweight *imlib2*-based *X11* image viewer that can be run from command-line arguments. You can find useful information about feh such as features, documentation, downloads, and how to build it from source at `https://feh.finalrewind.org/` (Figure 3-21):

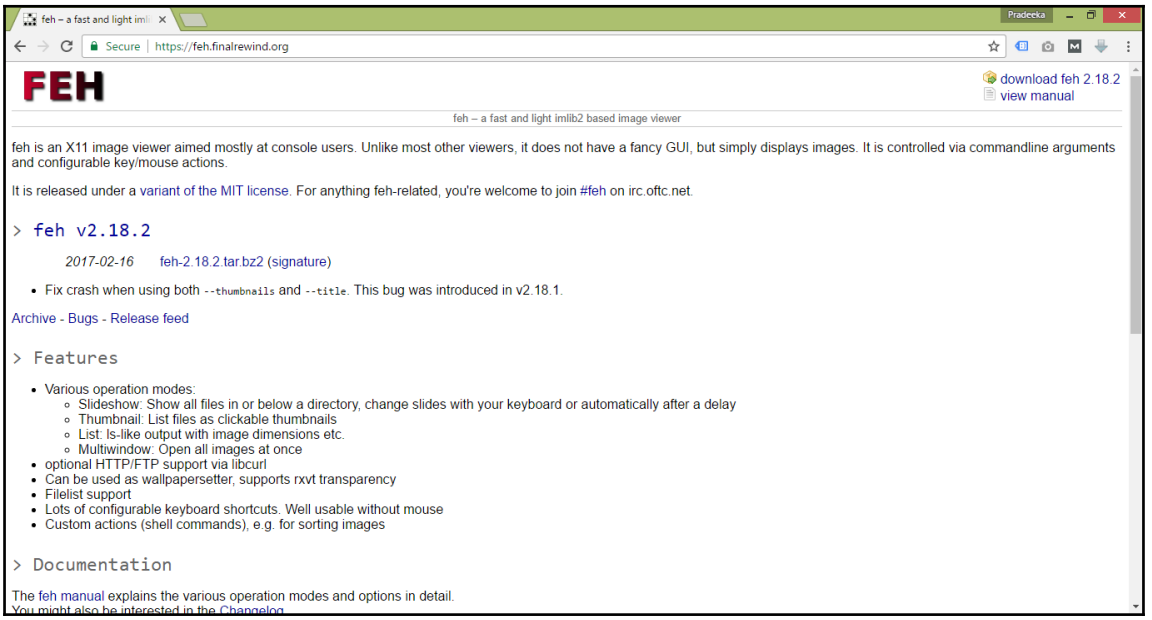


Figure 3-21: feh home page at `https://feh.finalrewind.org/`

The installation process is easy with a single command:

1. Open the Raspberry Pi terminal and run the following command. feh will be installesd on your Raspberry Pi within a few seconds:

```
sudo apt-get install feh
```

2. Now type the following command to display all the images in the pictures directory recursively at 5-second intervals. The command will refresh the source folder every 1 hour (3,600 seconds) and look for new images. The delay between images:

```
feh --recursive --slideshow-delay 3 --reload
3600 -F
/home/pi/RASPI3JAVA/DigitalPhotoFrame/media/
```

3. feh displays all the images recursively as a slideshow, as shown in the (Figure 3-22):

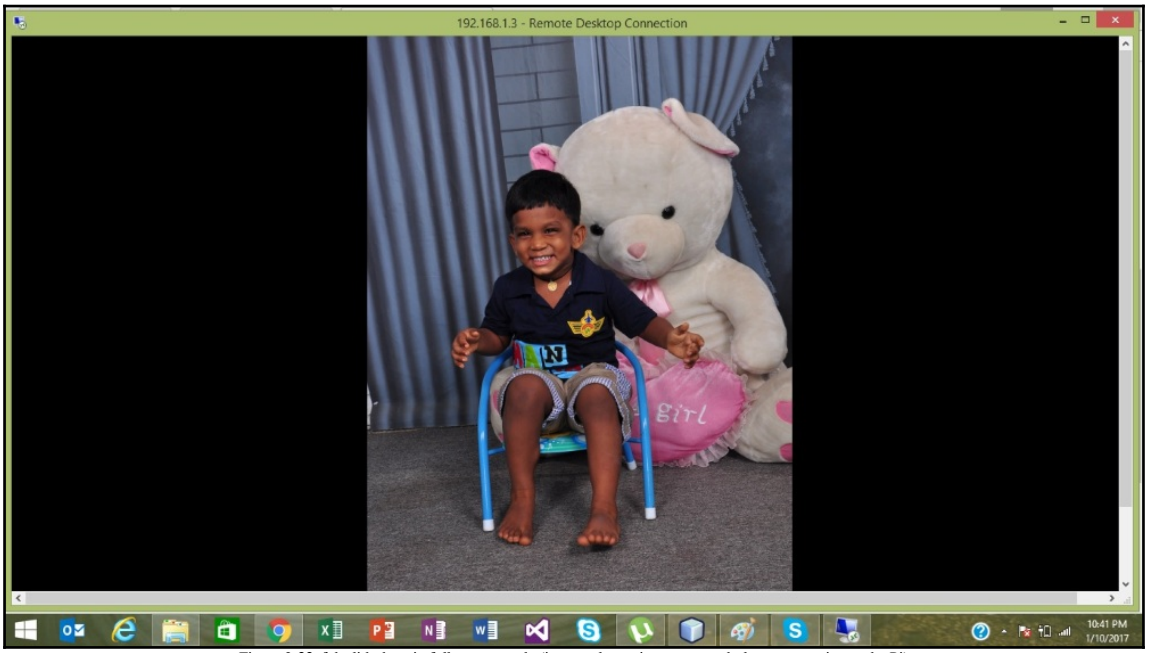


Figure 3-22: feh slideshow in full screen mode (image taken using a remote desktop connection to the Pi)

Scheduling your application

To keep the local image library up to date with the Flickr album, you should run your Java application regularly. The distribution file for your application is built every time you compile your application with NetBeans or the command line, with the use of the JDK. The executable `DigitalPhotoFrame.jar` file can be found in the `/home/pi/RASPI3JAVA/DigitalPhotoFrame/dist/` directory. The file can be executed from the console with the following command:

```
java –jar /home/pi/RASPI3JAVA/DigitalPhotoFrame/dist/DigitalPhotoFrame.jar
```

The Figure 3-23 shows the console output of the executed `.jar` file of your application. If you're getting an output similar to this, you are ready to proceed with the next step, which is writing a shell script:

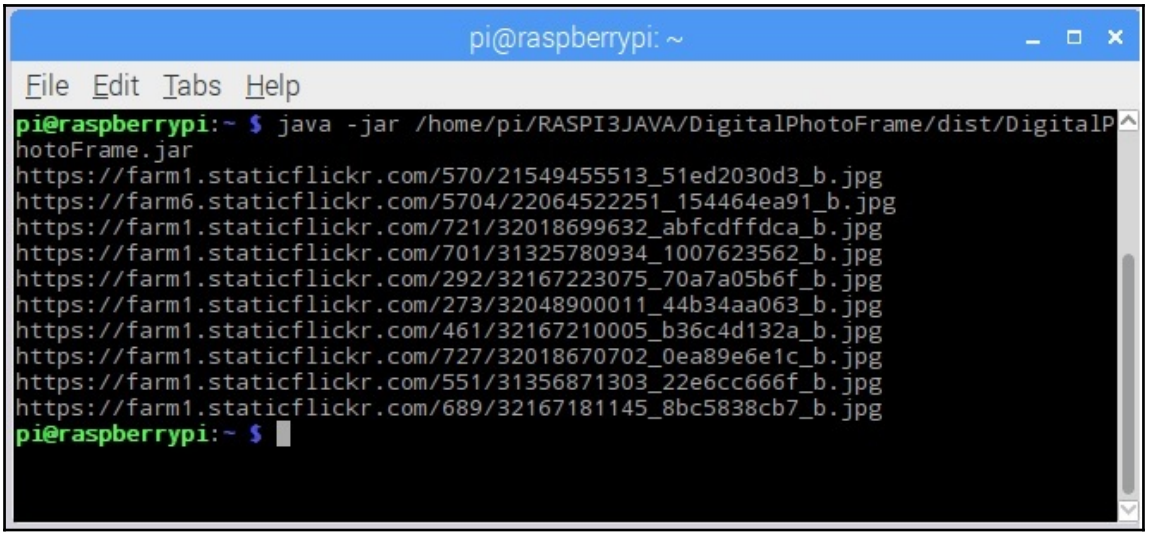


Figure 3-23: DigitalPhotoFrame.jar run in the terminal

Writing shell script for Java application

A simple shell script can be used to execute the `DigitalPhotoFrame.jar` application. Use the following steps to create a shell script with Raspbian OS:

1. Open a new text file named `digital_photo_frame.sh` by running this command:

```
sudo nano digital_photo_frame.sh
```

2. Now type the following lines in your file:

```
#!/bin/bash
java -jar
/home/pi/RASPI3JAVA/DigitalPhotoFrame/dist/
DigitalPhotoFrame.jar
```

3. Save the file by pressing *Ctrl* + *O*. Then press `Enter` to save the file with the same filename, `digital_photo_frame.sh`.
4. Then, exit the editor by pressing *Ctrl* + *X*.
5. To make the script executable, run the following command from the terminal:

```
sudo chmod +x digital_photo_frame.sh
```

Testing the digital_photo_frame.sh with the terminal

You can test the previous shell script by executing it with the Raspberry Pi terminal. Open the terminal and run the following command:

```
/home/pi/RASPI3JAVA/DigitalPhotoFrame/scripts/digital_photo_frame.sh
```

If you are in the scripts directory, you can run the script by simply using this command:

```
./digital_photo_frame.sh
```

The terminal will show an output similar to the Figure 3-24:

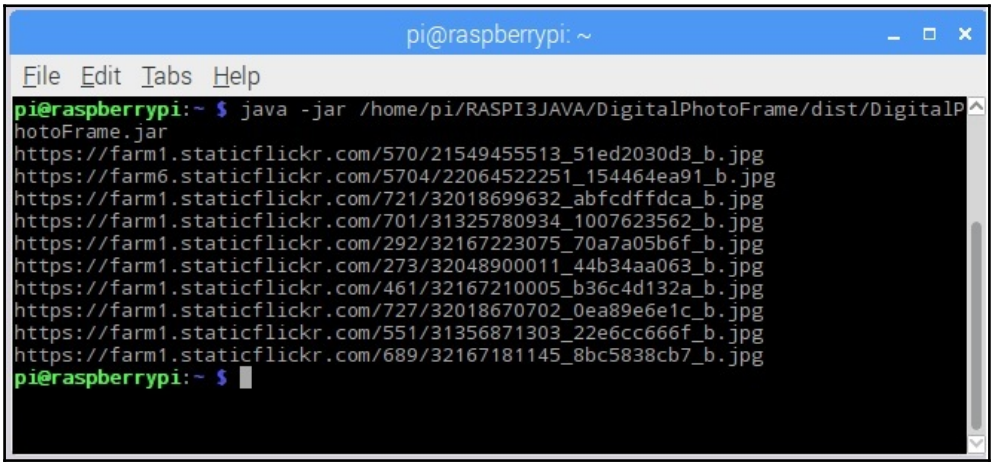


Figure 3-24: Terminal output for digital_photo_frame.sh

Now open the `/media` directory with **File Manager** to see the downloaded files.

If you're getting any errors, check the contents of `digital_photo_frame.sh` by opening the file using a text editor.

Scheduling digital_photo_frame.sh with crontab

Now you are ready to schedule your `digital_photo_frame.sh` file, which contains the execution instructions for the `.jar` file. C**ron** is a configuration tool used to run scheduled tasks on Raspberry Pi. Let's learn how to use crontab to schedule the shell script (Java application):

1. First, edit `crontab` by running the following command:

```
crontab -e
```

2. Choose nano as the editor by typing the option number and pressing *Enter.*
3. At the end of the file, type the following cron entry. This will run your `digital_photo_frame.sh` every day at midnight:

```
0 0 * * * java -jar
/home/pi/RASPI3JAVA/DigitalPhotoFrame/scripts/
digital_photo_frame.sh
```

4. The layout for a cron entry should be: minute, hour, day of month, month of year, day of week, and the command to be executed.
5. Save the file by pressing *Ctrl+O.*
6. Finally, exit crontab by pressing *Ctrl+X.*

Testing digital_photo_frame.sh with crontab

You can test the scheduled script without waiting for midnight by modifying the time of the cron entry to be very close to the current system time.

First, delete all the downloaded images in the `/media` directory.

For example, if your Raspberry Pi's system time is 10.30 A.M., modify the crontab entry by adding about 2 or 3 minutes to the first parameter of the entry, as shown:

```
32 10 * * * java -jar
/home/pi/RASPI3JAVA/DigitalPhotoFrame/scripts/digital_photo_frame.sh
```

Then, quickly save the file *(Ctrl + O)* and exit *(Ctrl +X)*. Wait until 10.32 am and browse the `/media` directory to see the downloaded images.

Writing shell script for slideshow

Let's write another simple shell script for the feh slideshow:

1. Open a new text file named `slideshow.sh` by running this command:

```
sudo nano slideshow.sh
```

2. Now type the following lines in your file:

```
#!/bin/bash
feh --recursive --slideshow-delay 3 --reload
3600 -F
/home/pi/RASPI3JAVA/DigitalPhotoFrame/media/
```

3. Save the file by pressing *Ctrl + O*. Then press *Enter* to save the file with the same filename, `slideshow.sh`.
4. Then, exit the editor by pressing *Ctrl + X*.
5. To make the script executable, run the following command from the terminal:

```
sudo chmod +x slideshow.sh
```

6. Now, test the `slideshow.sh` script by running the following command from the terminal:

```
/home/pi/RASPI3JAVA/DigitalPhotoFrame/scripts/slideshow.sh
```

7. The slideshow will start and display all the images in your `/media` directory.

Starting digital photo frame on Raspberry Pi boot

It is very important to automatically start your compiled Java application (`DigitalPhotoFrame.jar`) and the feh slideshow on boot. This can be done by editing the `rc.local` file. The following steps will show you how to edit the `rc.local` file to automate the digital photo frame on system boot:

1. Open the `rc.local` file by running the following command:

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

2. Scroll down to the bottom of the file. Add the following two lines just before `exit 0`:

```
                /home/pi/RASPI3JAVA/DigitalPhotoFrame/scripts/
                digital_photo_frame.sh &&
/home/pi/RASPI3JAVA/DigitalPhotoFrame/scripts/slideshow.sh &
```

3. Save the file by pressing *Ctrl* + *O*, followed by pressing *Enter.*
4. Exit from the editor by pressing *Ctrl* + *X*.
5. Now, run the following command to test the digital photo frame with `rc.local`:

```
/etc/rc.local
```

Photo frame in action

Now you have successfully built your digital photo frame, and installed and configured everything with the Raspberry Pi. Now reboot the Raspberry Pi. When the Pi boots up, the Java application will start to download photos from Flickr and feh will start the slideshow in full screen mode.

Summary

In this chapter, you've learned how to build a social and digital photo frame using the Raspberry Pi 3. Although we used Flickr as the social media platform, you can use other social media platforms as well, such as Facebook or Instagram. There are many ways to improve this project. For example, you could display comments along with the photos using the `flickr.photos.comments.getList` API method.

`Chapter 4`, *Integrating a Real-Time IoT Dashboard*, presents how to build a real-time IoT dashboard that can be used to display sensor data, and switch local actuators attached to the Raspberry Pi with Adafruit IO, MQTT, and the I2C serial communication protocol.

4

Integrating a Real-Time IoT Dashboard

The Raspberry Pi is great for working with sensor systems and controlling actuators in both indoor and outdoor environments. The data can be gathered from local sensors attached to the Raspberry Pi, or sensors connected to a cloud computing environment. Adafruit IO is a system that makes data useful, and it provides useful features to work with your data, such as real-time graphs and sets of UI controls that can be used to either display data from a feed or produce data for a feed on a specific topic.

In this chapter, you will:

- Create feeds and a dashboard with Adafruit IO
- Connect the TMP102 temperature sensor with the Raspberry Pi through an I2C bus
- Get data from the temperature sensor through the I2C with the `Pi4J` library
- Work with the `Eclipse Paho Java MQTT client` library
- Publish data to the feed on a specific topic
- Display data on the Adafruit IO dashboard
- Subscribe to a feed to get data on a specific topic
- Control an actuator using the Adafruit IO dashboard

Adafruit IO

Adafruit IO is a system that makes your data useful and provides easy-to-use, simple data connections. Basic programming is required to work with the Adafruit IO system, which includes client libraries that wrap its REST and MQTT APIs. Currently, Adafruit provides client libraries to work with following programming languages:

- Arduino
- Ruby
- Python
- Node.js

Still, there is no official `Adafruit client` library to work with Java. So, you can use the Eclipse Paho MQTT Java client to work with Adafruit IO.

You can visit Adafruit IO at `https://io.adafruit.com/`.

Bill of materials

To build all the projects that will be discussed in this chapter, you will need to have the following things in your tool box:

- One Raspberry Pi 3 (SparkFun part number: DEV-13825)
- One SparkFun digital temperature sensor breakout board - TMP102 (SparkFun part number: SEN-11931)
- One LED
- One breadboard-self-adhesive (white) (SparkFun part number: PRT-12002)
- Hookup wire or jumper wire kit
- Wire strippers

Sign in with Adafruit IO

You will need an Adafruit user account to sign in to Adafruit IO:

1. Using your web browser, visit `https://io.adafruit.com/` and click the large text button, **SIGN IN TO JOIN THE OPEN BETA**.
2. On the next page, there are two options available for you. If you already have an account with Adafruit, you can use the same credentials to sign in with Adafruit IO by giving the credentials (email/username and password) and clicking on the **SIGN IN** button. If you don't have an account, click the **SIGN UP** button to create a new account.

After successfully signing in to Adafruit IO, you will navigate to the Adafruit IO dashboards page, which includes a list of dashboards. Figure 4-1 shows a section of the Adafruit IO dashboards page. The dashboards page is generally located at the (your user name)/dashboards path:

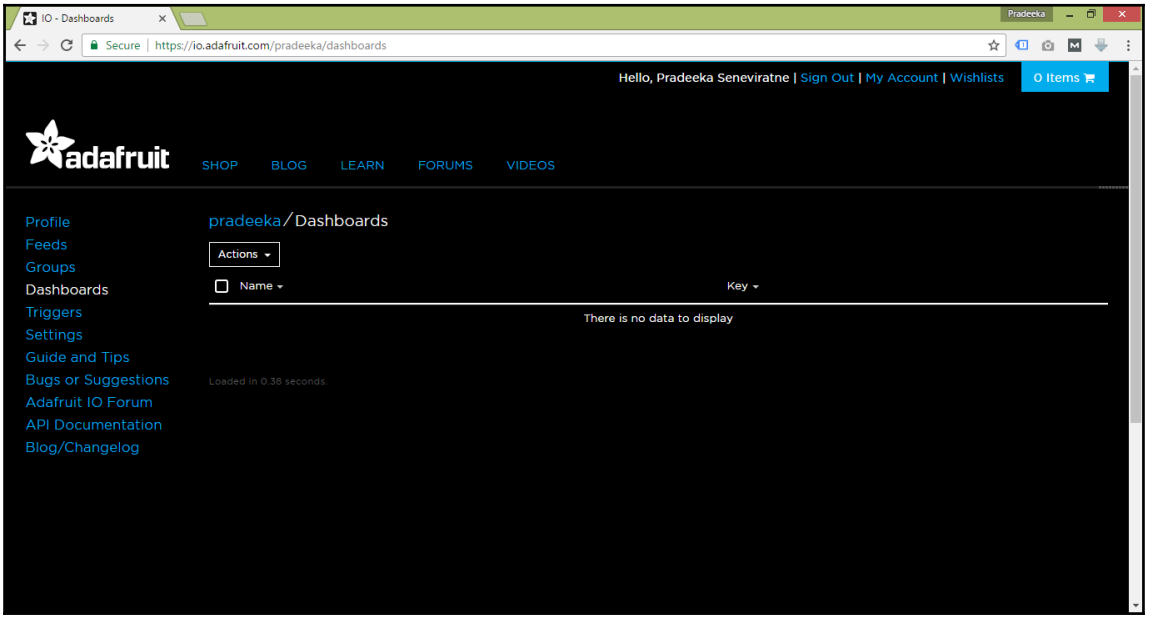


Figure 4-1: Adafruit IO dashboards page

3. A navigation menu is located on the left-hand side of the page, and provides links to your profile, feeds, groups, dashboards, triggers, and settings.

Finding your AIO key

The Adafruit (AIO) IO key is used to authenticate your client connection with the Adafruit IO system:

1. In the left navigation bar of the dashboards page, click the Settings menu. You will navigate to the Settings page with a set of settings associated with your Adafruit IO account, as shown in Figure 4-2:

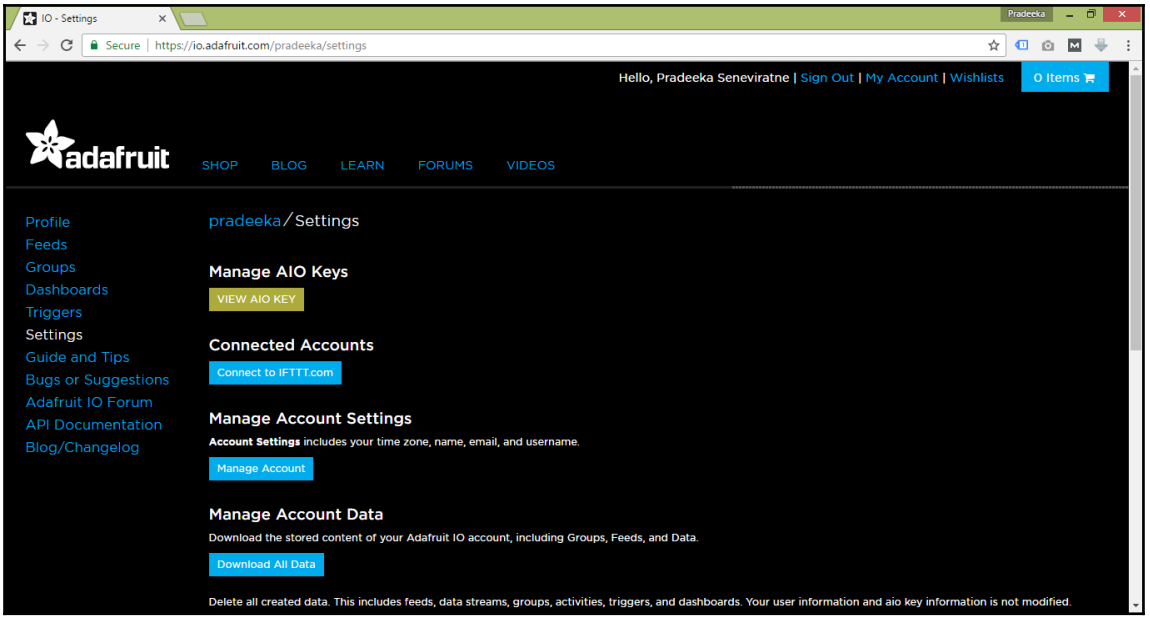


Figure 4-2: Adafruit IO settings page

2. Click **VIEW AIO KEY** under **Manage AIO Keys**. A pop-up window will show the AIO key associated with your account, as shown in Figure 4-3:

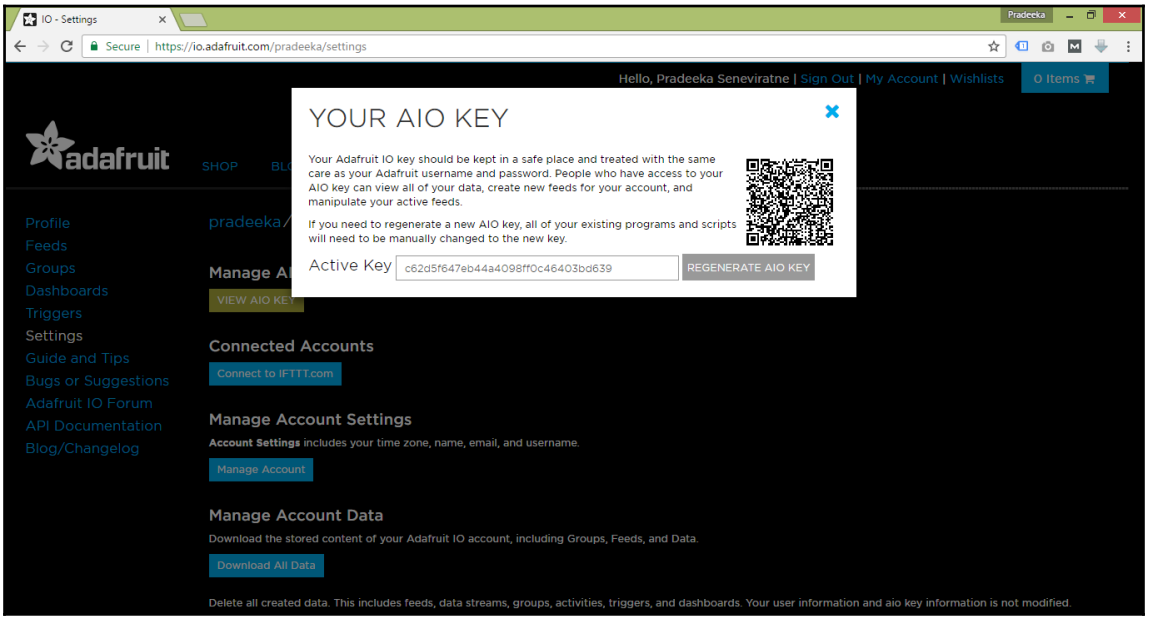


Figure 4-3: Adafruit AIO key

The 32-digit AIO key looks similar to this:

c62d5f647eb44a4098ff0c46403bd639

If you need to regenerate a new AIO key, click **REGENERATE AIO KEY** button. After generating a new AIO key, your current applications no longer work with the old API key, so replace them with the new AIO key.

Creating news feed

A feed contains data and metadata about values that get pushed to Adafruit IO. This data can be populated through REST API, Python, Ruby, and Node.js clients. In this chapter, you will use the Eclipse Paho MQTT Java client to populate data.

The following steps will help you to create a new feed for the temperature data that's generated by the TMP102 temperature sensor.

Adafruit IO is currently in beta version at the time of writing this. Intermittently, you will encounter unpredictable behaviors when you are working with Adafruit IO.

1. In the left navigation bar, click the **Feeds** menu. You will navigate to the **Feeds** page located at the (username)/feeds path, as shown in Figure 4-4:

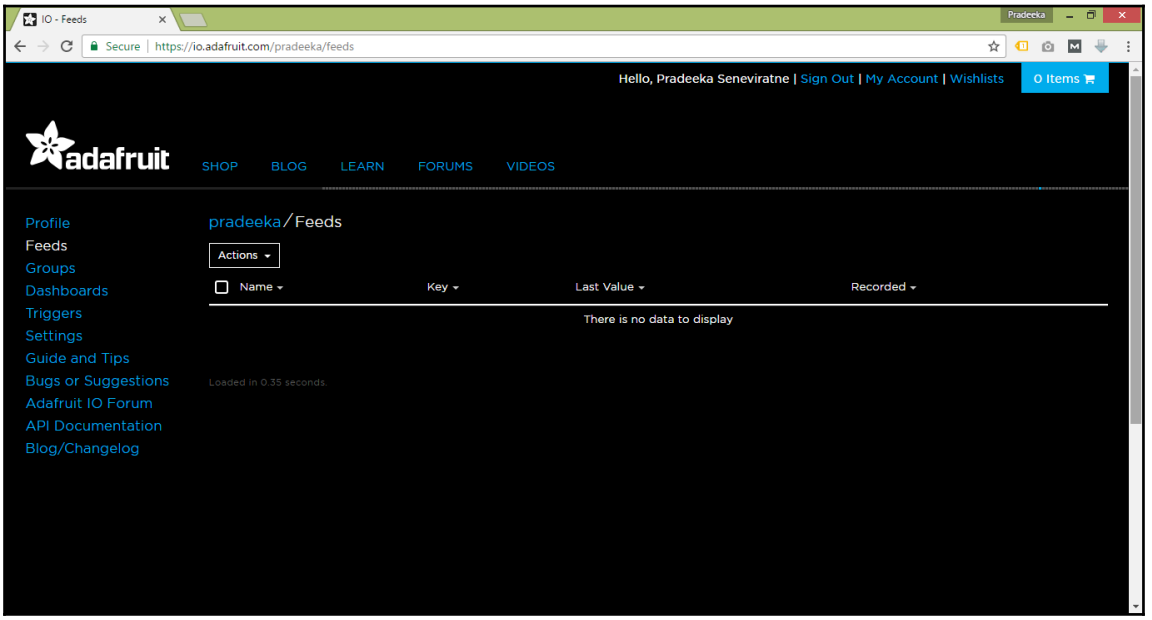


Figure 4-4: Adafruit IO Feeds page

2. Click the **Actions** button, and from the drop-down menu, click **Create a New Feed** as shown in Figure 4-5:

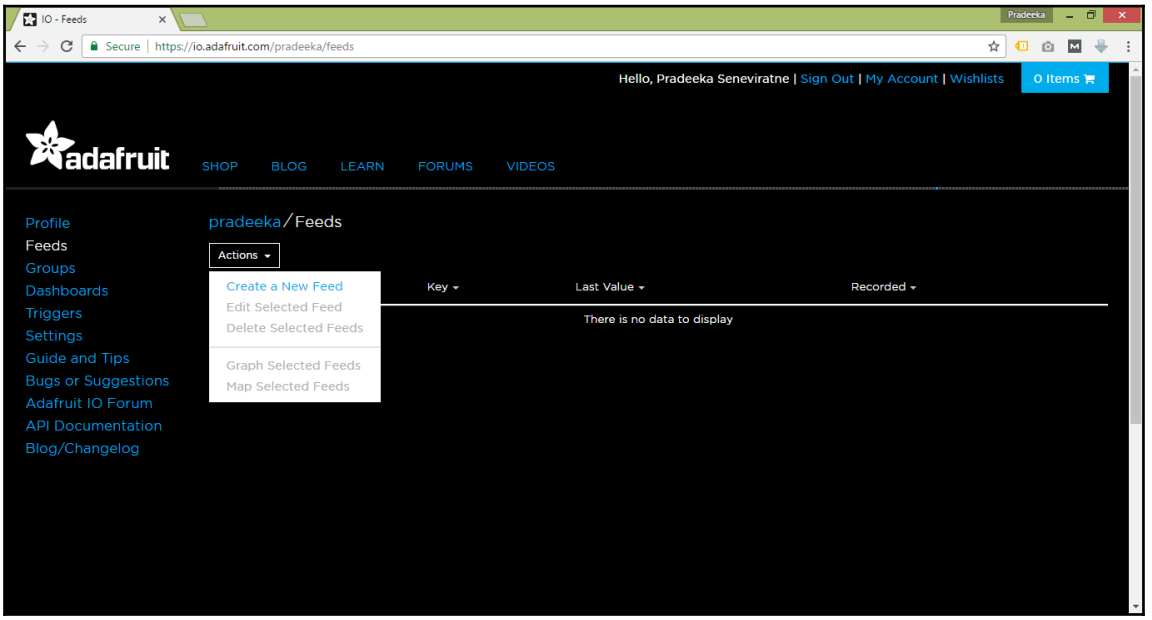


Figure 4-5: Menu selection for Create a new Feed

3. The **Create a new Feed** dialog box will pop up. Type the following information for each text box and click the **Create** button, as shown in Figure 4-6:
    - **Name:** `Temperature`
    - **Description:** `TMP102 Temperature Sensor`

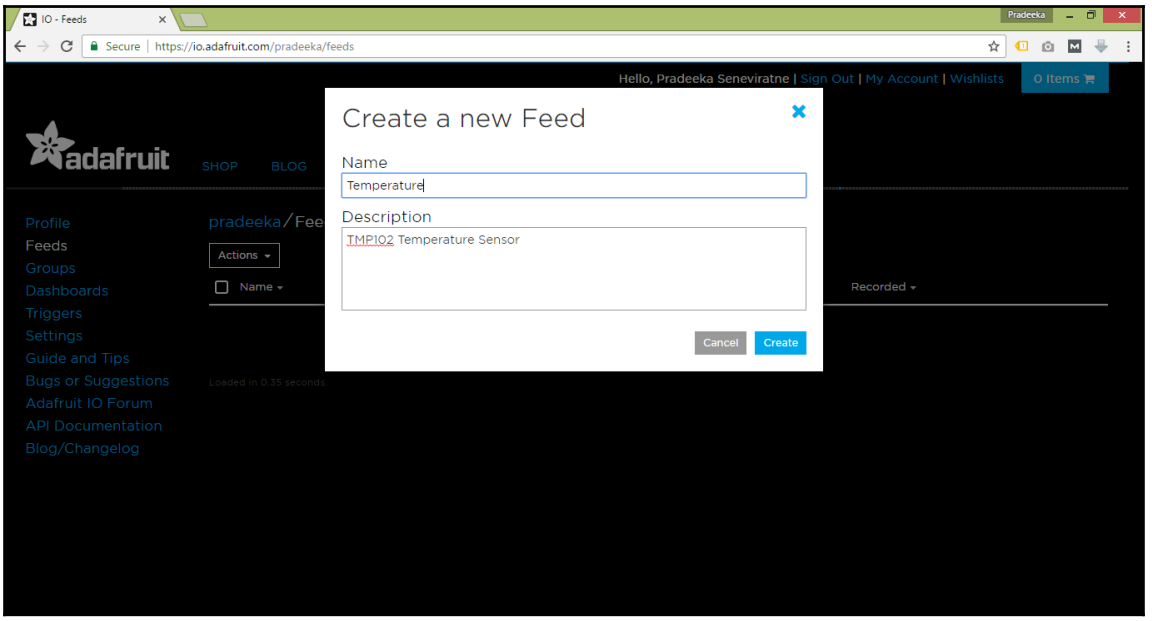


Figure 4-6: Create a new Feed dialog box

4. The newly created temperature feed is now listed on the Feeds page, as shown in the Figure 4-7:

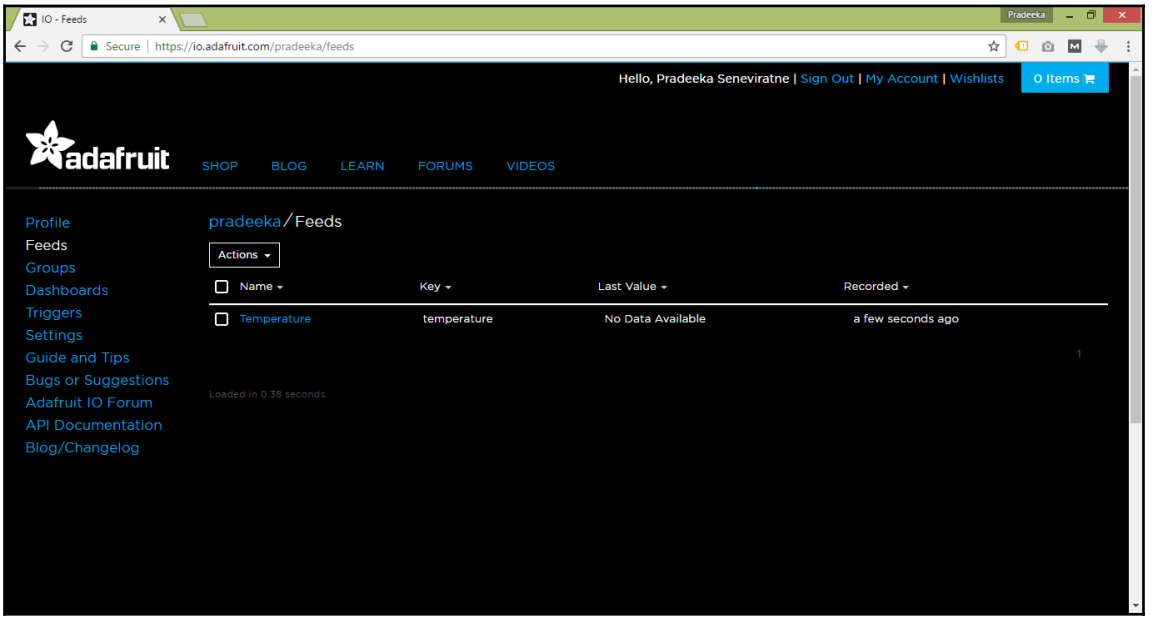


Figure 4-7: Temperature feed added to the Feeds list

5. In the list, click on the **Temperature** feed (feed **Name**) to view the feed data. Currently, you will get a blank graph with the message `There is no data to display` because you haven't published data to the feed yet (Figure 4-8):

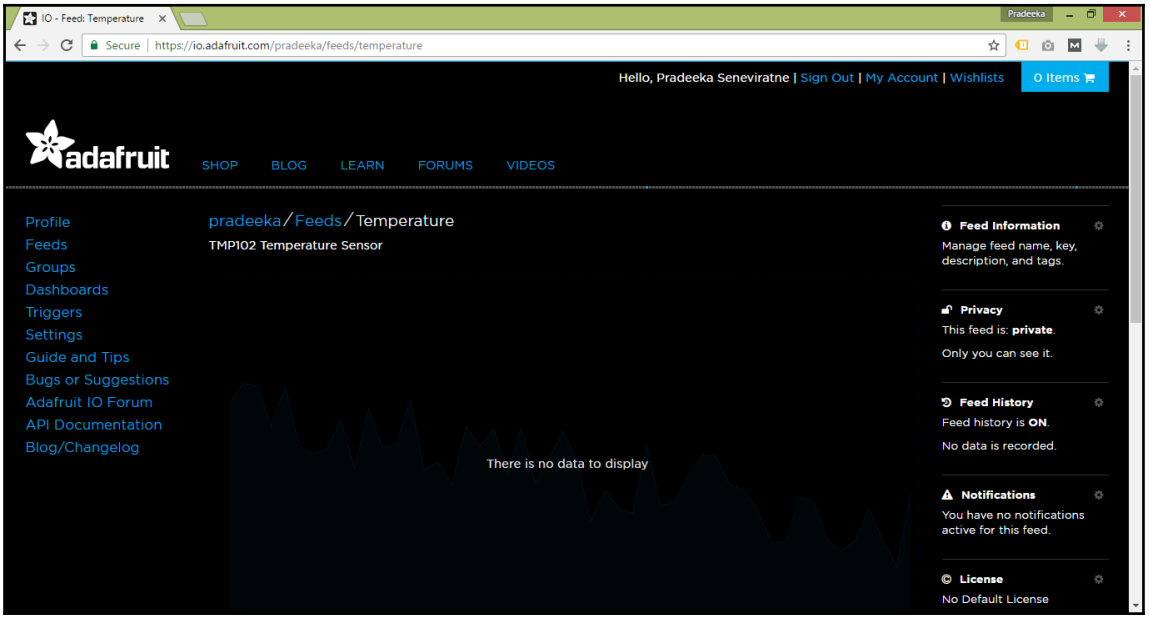


Figure 4-8: Display page for feed data

Understanding topics

using its topic. A topic can have one of the following forms, where username is your Adafruit account username:

You can publish a new value for a feed using its topic. A topic can have one of the following forms, where username is your Adafruit account username: (username)/feeds/(feed name or key) (username)/f/(feed name or key)

The topic for the *Temperature* feed can be written as one of the following:

Name-based:

```
pradeeka/feeds/Temperature
 pradeeka/f/Temperature
```

Key-based:

```
pradeeka/feeds/temperature
 pradeeka/f/temperature
```

Creating a dashboard

1. In the left navigation bar, click on the **Dashboards** menu. This will bring you to the **Dashboards** page located at the (username)/dashboards path, as shown in Figure 4-9:

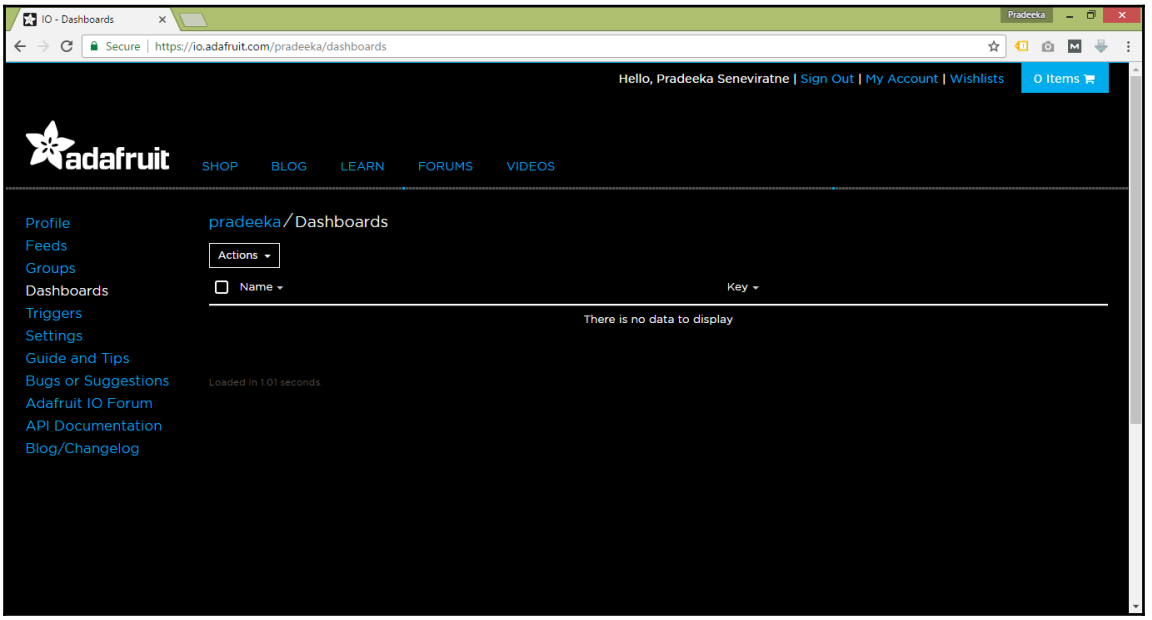


Figure 4-9: Dashboards page

2. Click on the **Actions** button and from the drop-down list, click **Create a New Dashboard**, as shown in Figure 4-10:

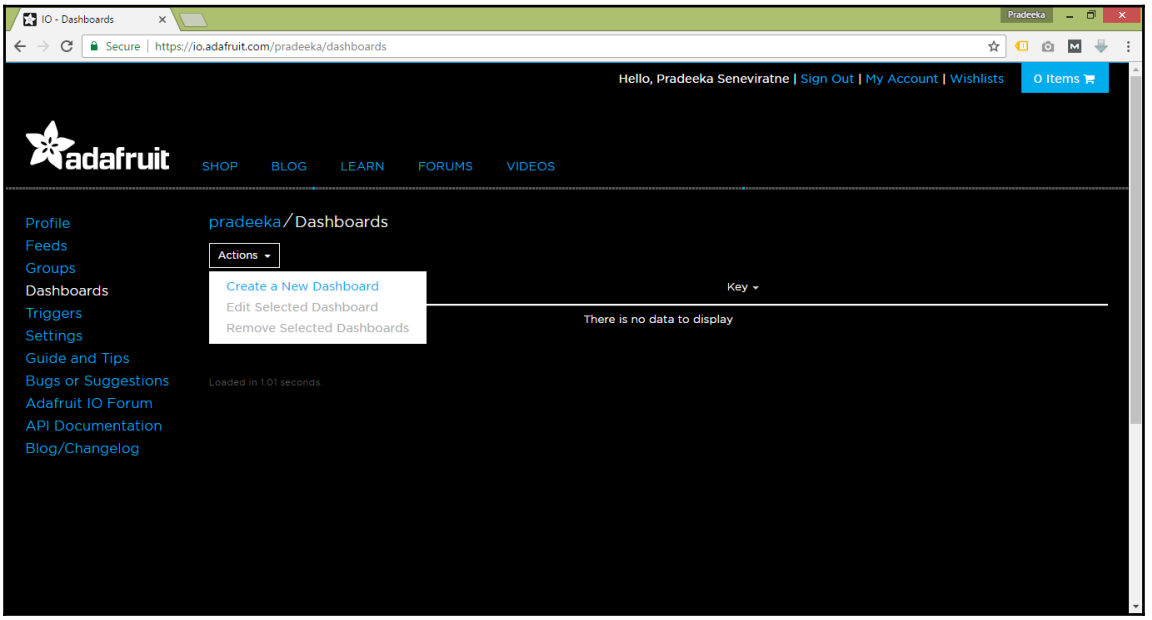


Figure 4-10: Menu selection for create a new dashboard

3. A pop-up dialog box, **Create a New Dashboard,** will appear (Figure 4-11) and prompt for the **Name** and **Description**. Type the following and click the **Create** button:
    - **Name:** `IoT Dashboard`
    - **Description:** `IoT Dashboard`

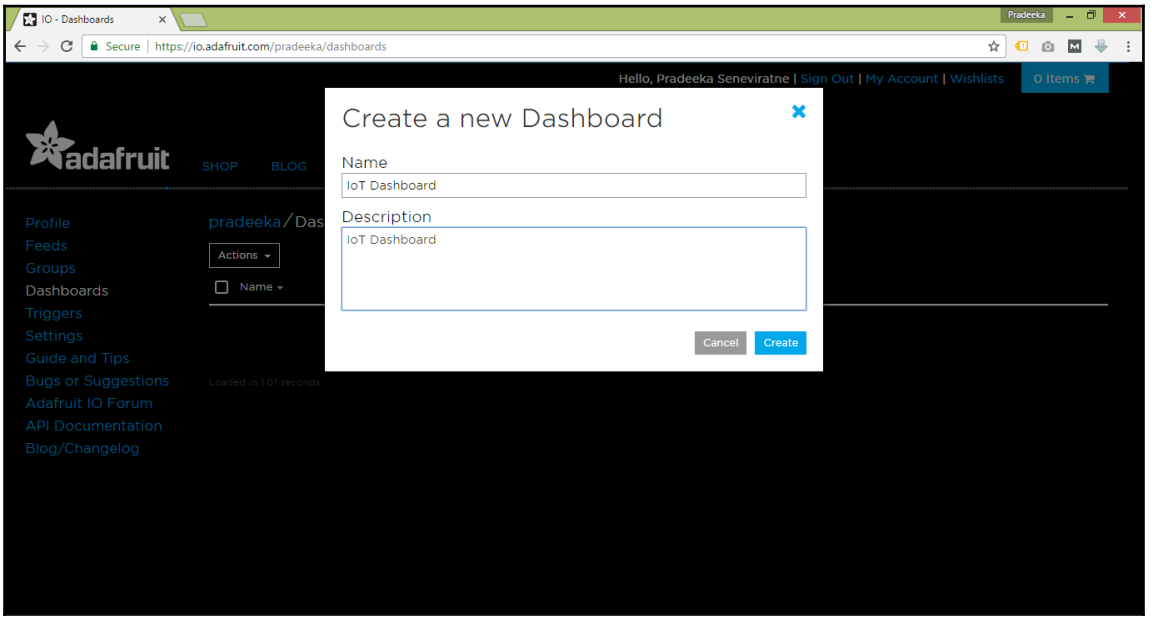


Figure 4-11: Create a new dashboard dialog box

4. The newly added dashboard will show in the dashboards list shown in Figure 4-12:

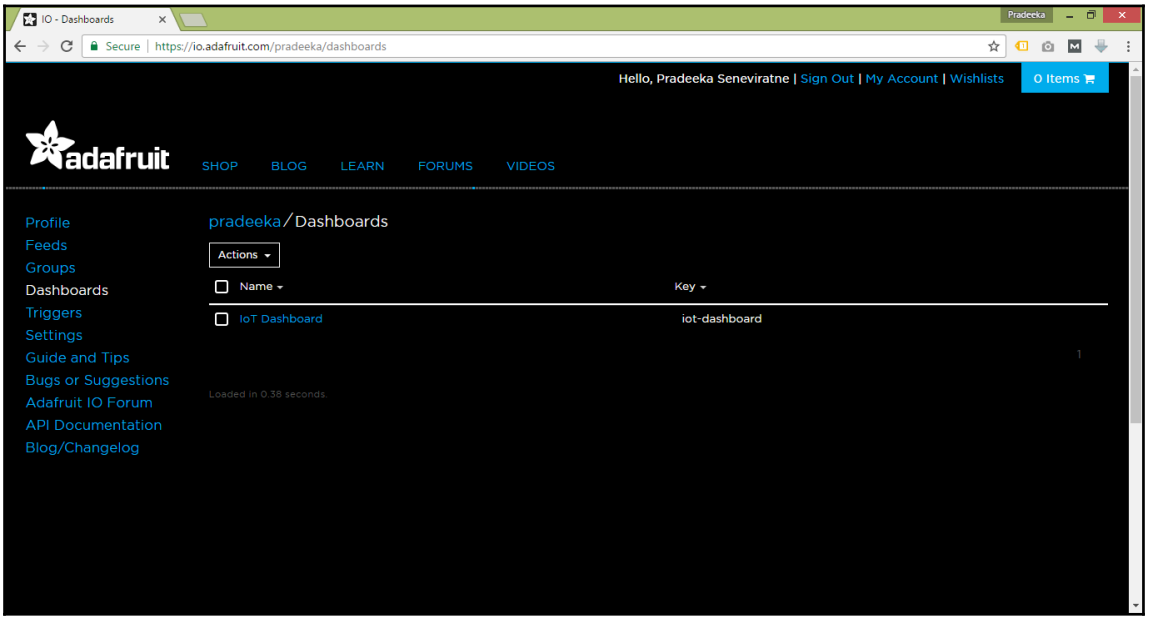


Figure 4-12: List of dashboards

Now you have successfully created a dashboard on the Adafruit IO system, which is associated with your account.

Creating a block on a dashboard

To display data on the dashboard, first you need to create a block to display data from a feed. Also, some blocks can be used to create data for a feed. The following blocks are currently available to use on the dashboard:

- Toggle
- Memory button
- Slider
- Gauge
- Text
- Stream

- Image
- Line Chart
- Color picker
- Map

The following steps will explain how to add a Line Chart block to your dashboard.

1. Click on the **IoT Dashboard** link under the **Name** column. You will navigate to the dashboard page located at the `(username)/dashboards/iot-dashboard` path, as shown in Figure 4-13:

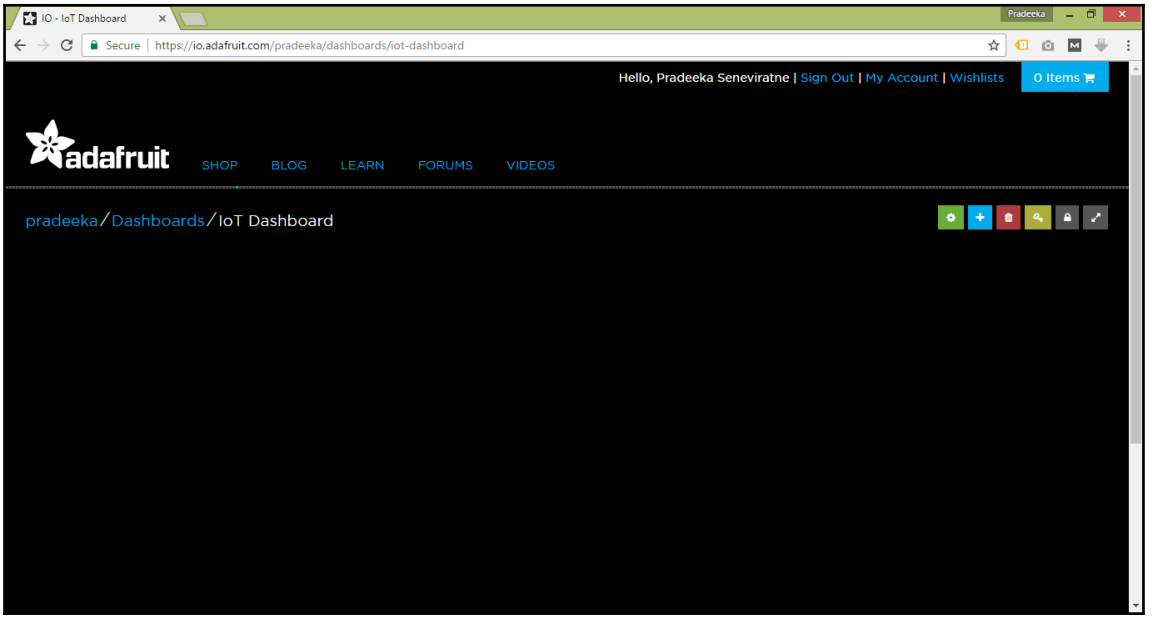


Figure 4-13: IoT dashboard page

2. Click the **Create a new block** button in the top-right corner of the page. The **Create a new block** pop-up window will appear (Figure 4-14). Click the **Line Chart** box:

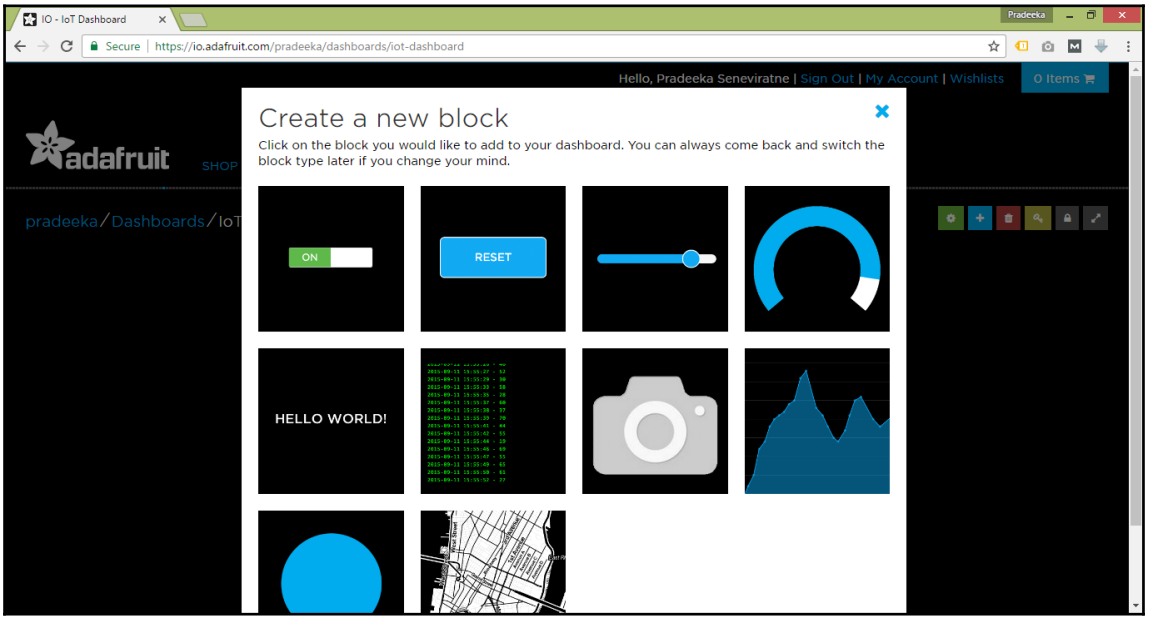


Figure 4-14: Create a new block

3. In the **Choose up to 5 feeds** dialog box, select **Temperature** under **Group/Feed** column and click the **Next step>** button (Figure 4-15):

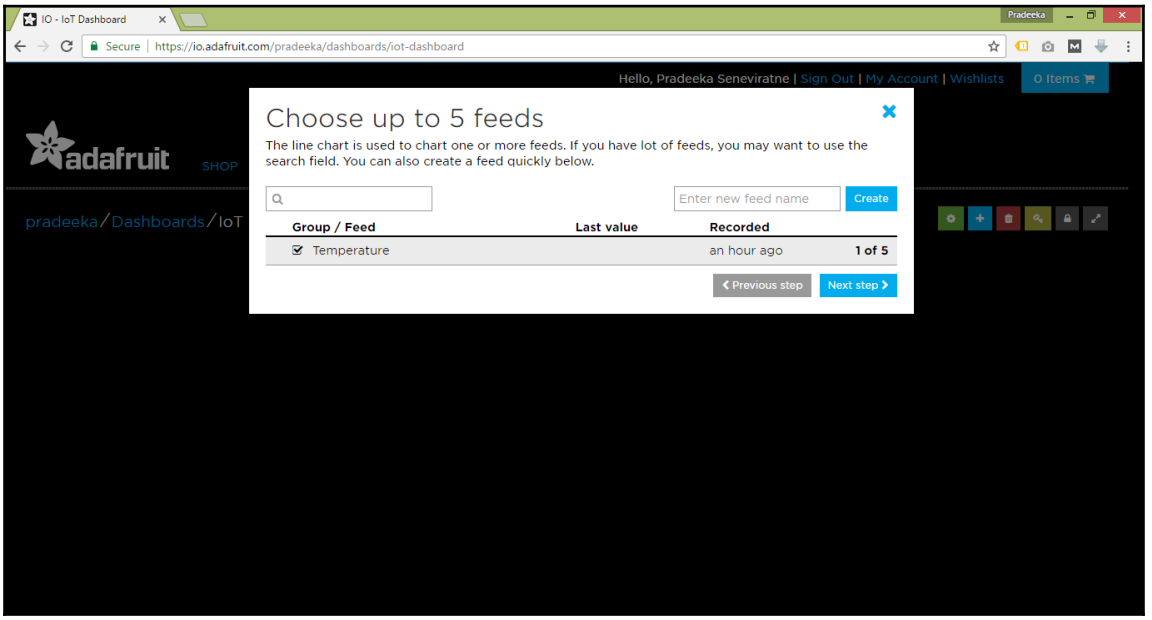


Figure 4-15: Selecting a feed

4. In the **Block settings** dialog box (Figure 4-16), type the following labels for the *X*-axis and *Y*-axis:
    - **X-Axis label**: `Time`
    - **Y-Axis label**: `Temperature (Celsius)`

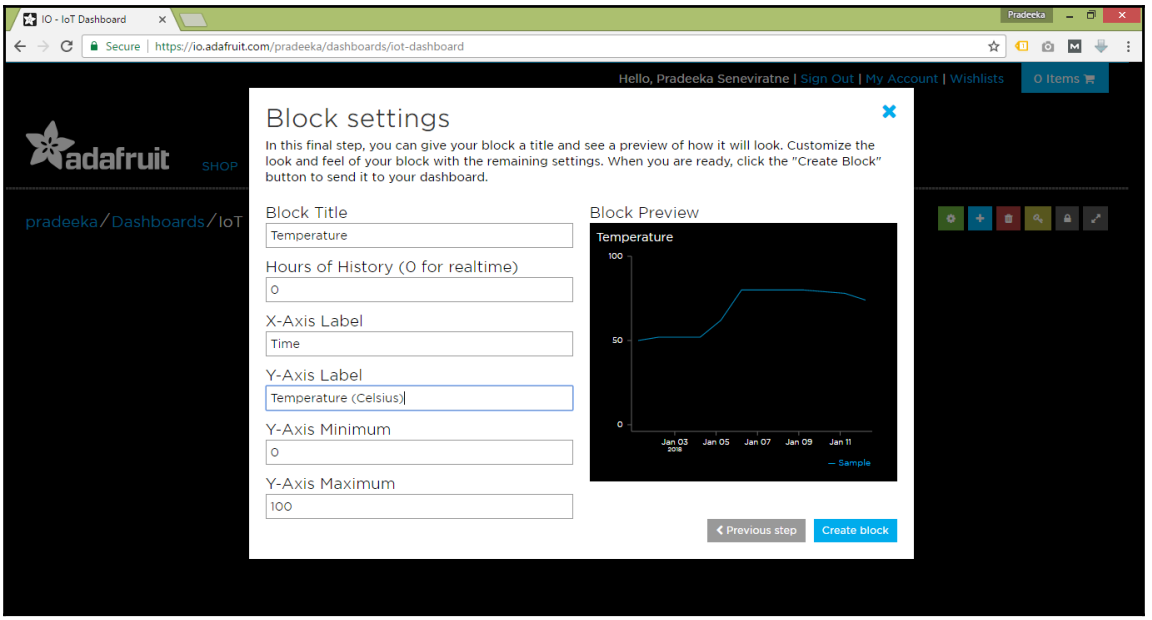


Figure 4-16: Block settings for Line Chart

5. A blank **Line Chart** block will be added to the IoT dashboard, as shown in Figure 4-17:

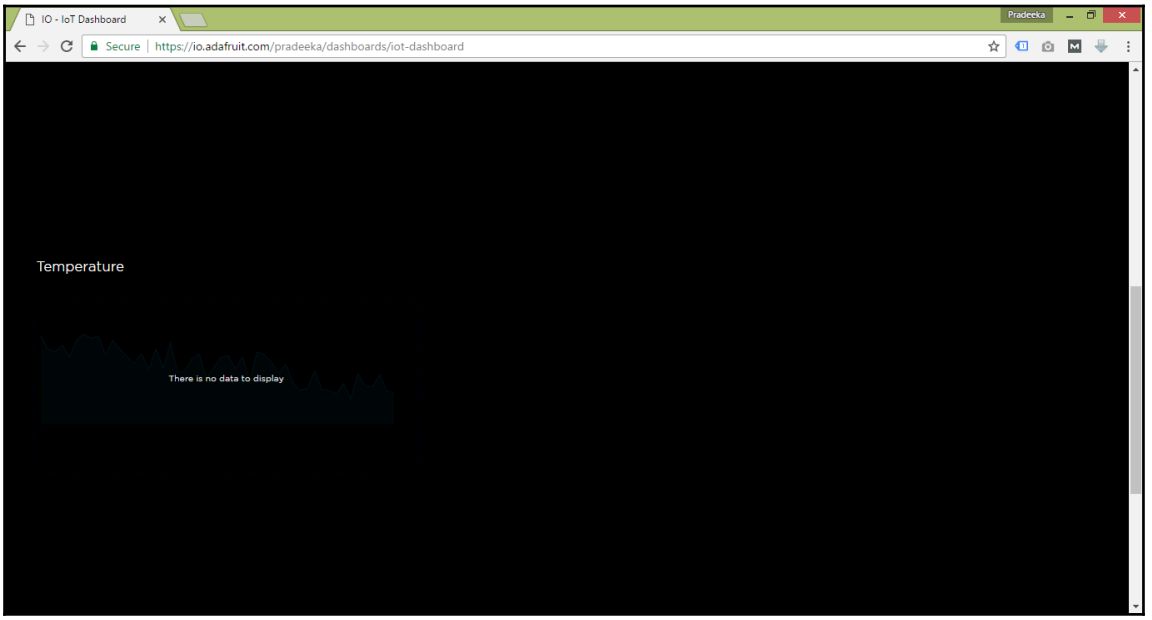


Figure 4-17: Line Chart bock with no data

Now you have successfully created a Line Chart block on the dashboard to display TMP102 temperature sensor data. The Line Chart block is currently empty because we haven't published data to the **Temperature** feed yet.

Raspberry Pi and I2C pins

Raspberry Pi has two I2C serial buses labeled **I2C.0** and **I2C1**. The two I2C interfaces of the Raspberry Pi can be accessed through the pins located in the GPIO header.

I2C.0 is disabled by default. To enable it, you'll need to manually edit the configuration file.

Figure 4-18 shows the Raspberry Pi pins associated with each I2C serial bus:

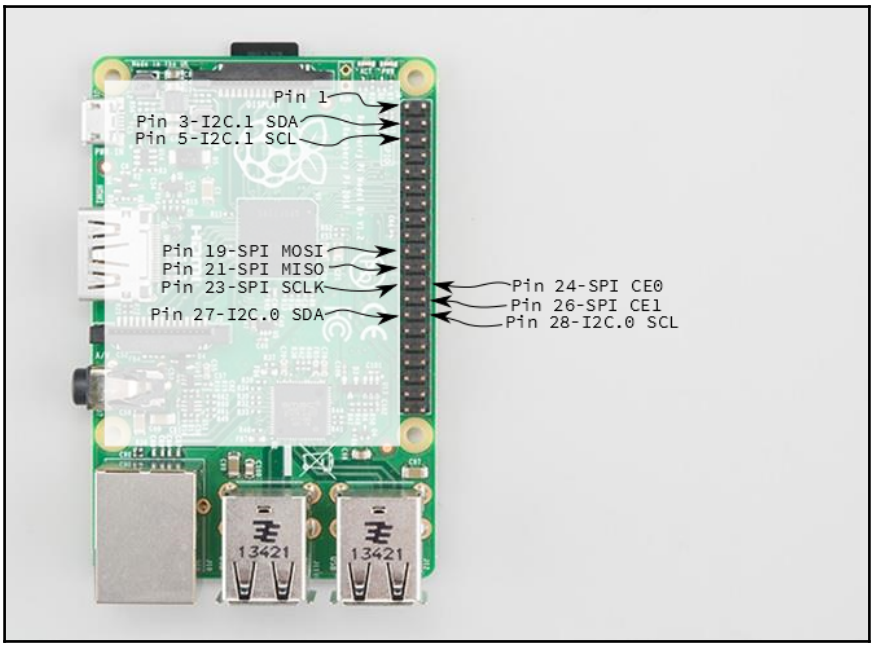


Figure 4-18: I2C serial bus pins. Image from SparkFun electronics (`https://www.sparkfun.com/`). Photo taken by Juan Peña

As you can see, the arrangement of the I2C pins is a little disorganized, as **I2C.1** is near one end, while **I2C.0** is in the middle of the header. To organized your I2C pins in a well-ordered way and use them with this project, purchase a Pi wedge adapter PCB from SparkFun electronics. You can find more information about the Pi wedge at `https://www.sparkfun.com/products/13717`. The kit includes a pre-assembled SparkFun Pi wedge and a GPIO ribbon cable (40-pin, 6"), which can be easily used with a breadboard to hook up your projects.

Connecting an I2C-compatible sensor to the Raspberry Pi

To play with I2C on your Raspberry Pi, you can purchase a cheap temperature sensor breakout board that is capable of I2C communication. The SparkFun digital temperature sensor breakout - TMP102 (SEN-11931) is a good choice to get started with I2C (Figure 4-19):

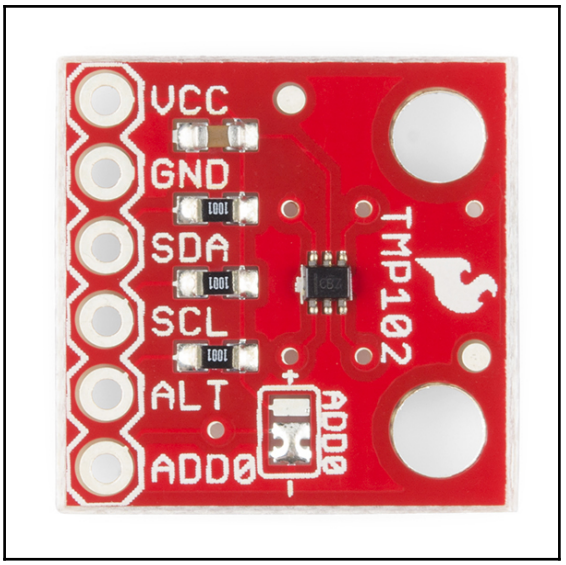


Figure 4-19: TMP102 temperature sensor breakout. Image from SparkFun Electronics (`https://www.sparkfun.com/`). Photo taken by Juan Peña

The heart of this breakout board is the TMP102 digital temperature sensor shown in Figure 4-20. It supports an I2C interface that only requires two wires to communicate with the Raspberry Pi:

Figure 4-20: TMP102 digital temperature sensor. Image courtesy of Texas Instruments (`http://www.ti.com/product/TMP102`)

The TMP102 digital temperature sensor breakout board has six connection pads:

- **VCC**: Positive power
- **GND**: Negative power
- **SDA**: To connect with the DATA line of the I2C bus
- **SCL**: To connect with the CLOCK of the I2C bus
- **ALT**: Unused
- **ADD0**: Connect to the ground

The supply voltage for the TMP102 digital temperature sensor breakout is between 1.4V and 3.6V DC.

The datasheet for the TMP102 can be found at `https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf`and is available as a downloadable file in PDF format. Download and save this file to your local drive for further reference.

Serial bus addresses

The TMP102 features an address pin (ADD0) to allow up to four devices to be addressed on a single bus as slave devices. Table 4-1 shows the slave addresses used to properly connect up to four devices:

Device two-wire address	**ADD0 pin connection**
1001000	Ground
1001001	V+
1001010	SDA
1001011	SCL

Table 4-1: Selection of slave addresses

The ADD0 pin of the TMP120 sensor should be connected to either the GND, VCC, SDA, or SCL pad, in order to assign an address to the TMP120 sensor before connecting it to the Raspberry Pi.

By default, the ADD0 pad of the TMP120 breakout board is solder-bridged to the GND (Figure 4-21) to assign the slave address 1001000:

Figure 4-21: ADD0 bridged with ground

Figure 4-22 shows the complete circuit diagram that can be used to get sensor data from the TMP120 temperature sensor:

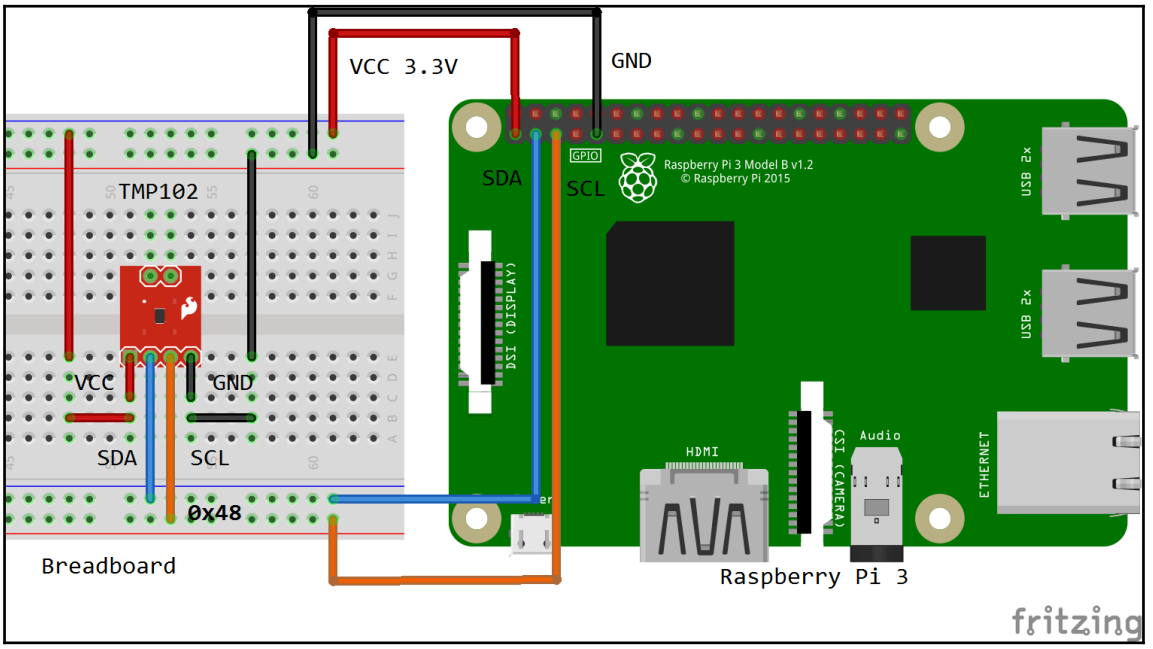


Figure 4-22: TMP120 sensor connected to Raspberry Pi circuit; Fritzing representation

You are now ready to create your first I2C connected sensor system. Use the following steps to connect the TMP102 breakout board to the Raspberry Pi:

1. Connect the VCC of the TMP102 to Raspberry Pi pin 1. This is the 3.3V DC Power pin.
2. Connect the GND of the TMP102 to Raspberry Pi pin 9. This is the Ground pin.
3. Connect the SDA of the TMP102 to Raspberry Pi pin 3. This is the GPIO 8 SDA1 (I2C) pin.
4. Connect the SCL of the TMP102 to Raspberry Pi pin 5. This is the GPIO 9 SCL1 (I2C) pin.
5. Now power your Raspberry Pi by connecting a 5V/2 A wall wart.

The Raspberry Pi will boot and connect with your Wi-Fi network (or Ethernet) automatically as you configured it in `Chapter 1`, *Setting up Your Raspberry Pi*.

Configuring the Raspberry Pi to use I2C

To use the I2C.1 interface of the Raspberry Pi with any I2C-enabled chip or device, you should first configure it with the Raspberry Pi. There are several ways to do this, but use the following steps to do it in an easy way:

1. Connect to your Raspberry Pi using PuTTY through SSH. See `Chapter 1`, *Setting up Your Raspberry Pi* for more information about how to use PuTTY.
2. Using the PuTTy terminal, run this command:

```
sudo raspi-config
```

3. Select **Advanced Options** from the Configuration Tool window (Figure 4-23) and press *Enter*:

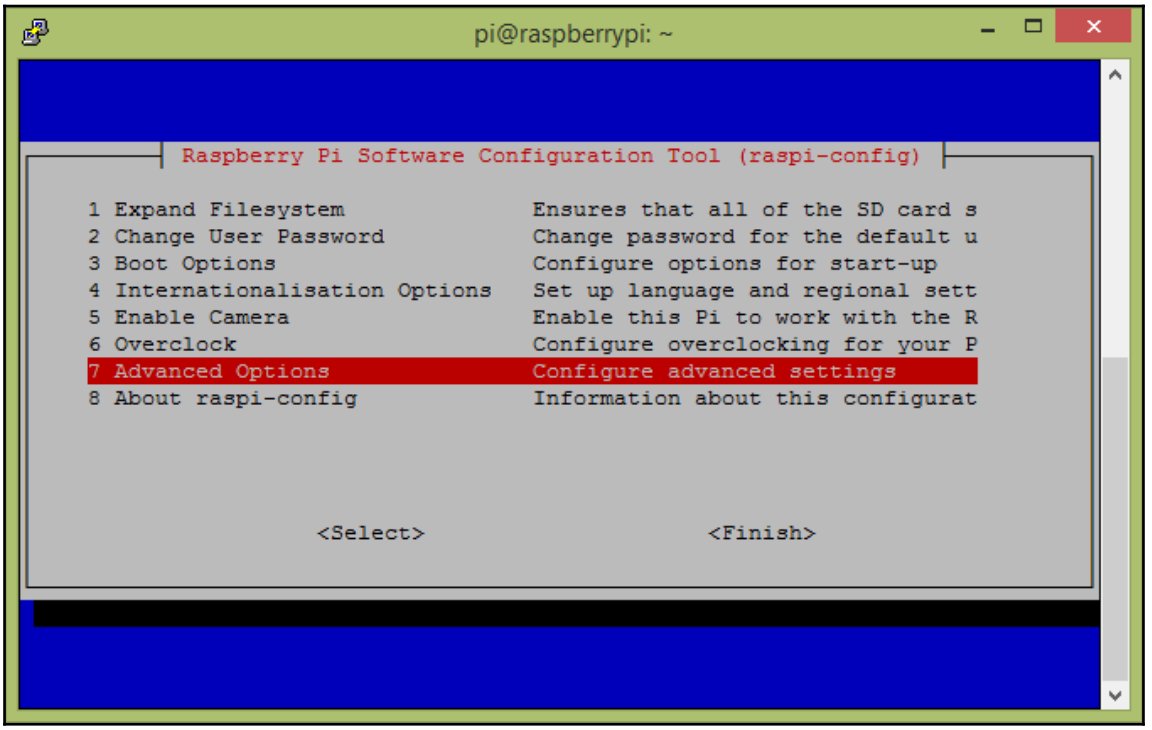


Figure 4-23: Selecting advanced options from the configuration tool window

4. In next window, select the **A7 I2C** option and press *Enter* (Figure 4-24):

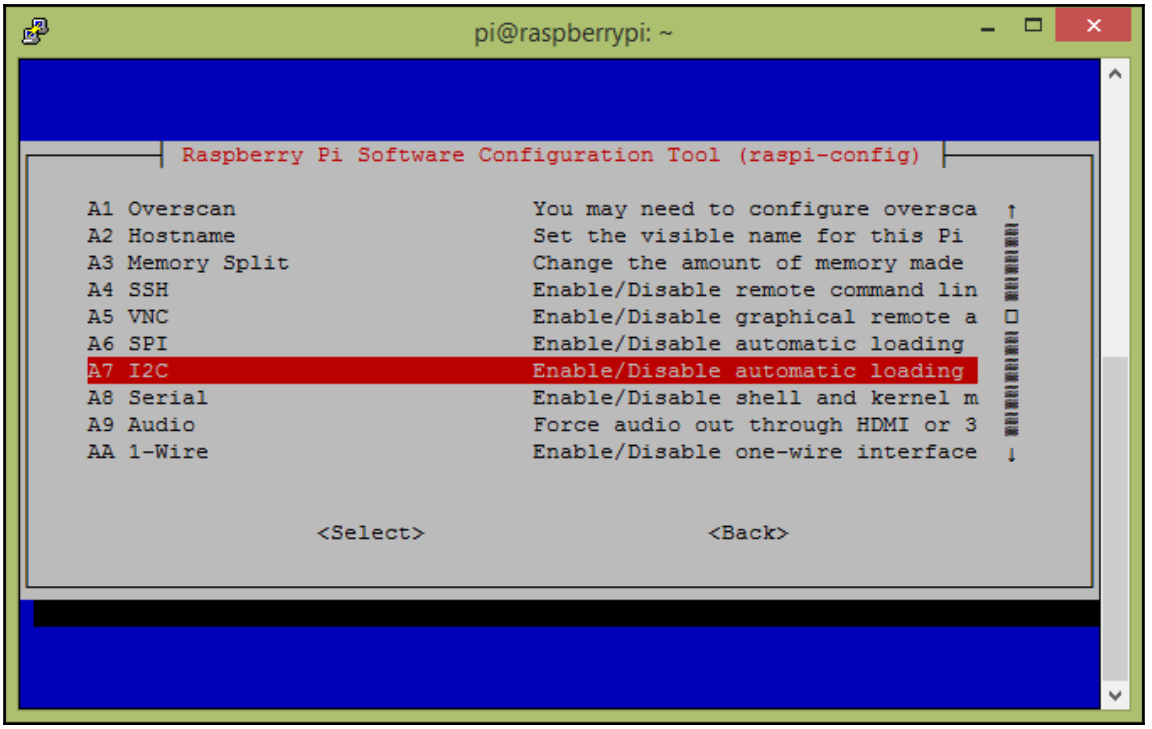


Figure 4-24: Selecting I2C from the configuration tool window

5. Select **<Yes>** and press *Enter* to enable the I2C interface on the Raspberry Pi (Figure 4-25):

Figure 4-25: Configuration tool window

6. A confirmation message will appear, saying **The ARM I2C interface is enabled** (Figure 4-26). Press *Enter* to close the message box:

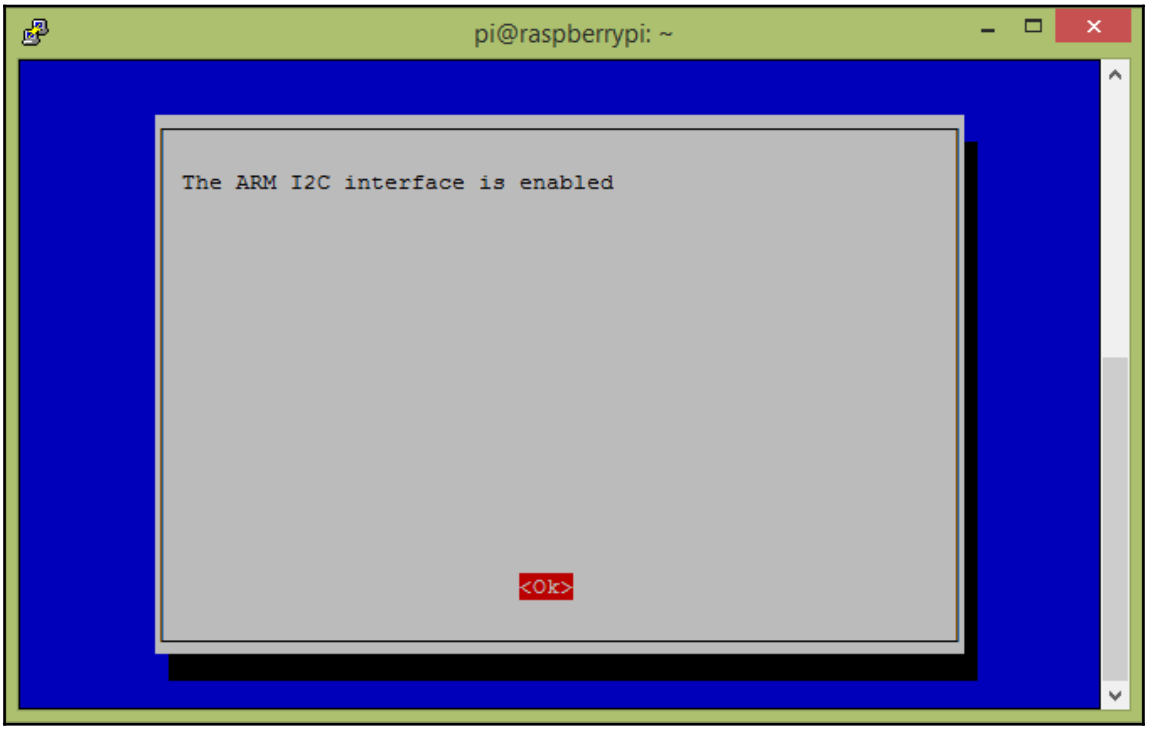


Figure 4-26: Configuration tool window

7. Select **<Finish>** and press *Enter* to exit the **Configuration Tool** window (Figure 4-27):

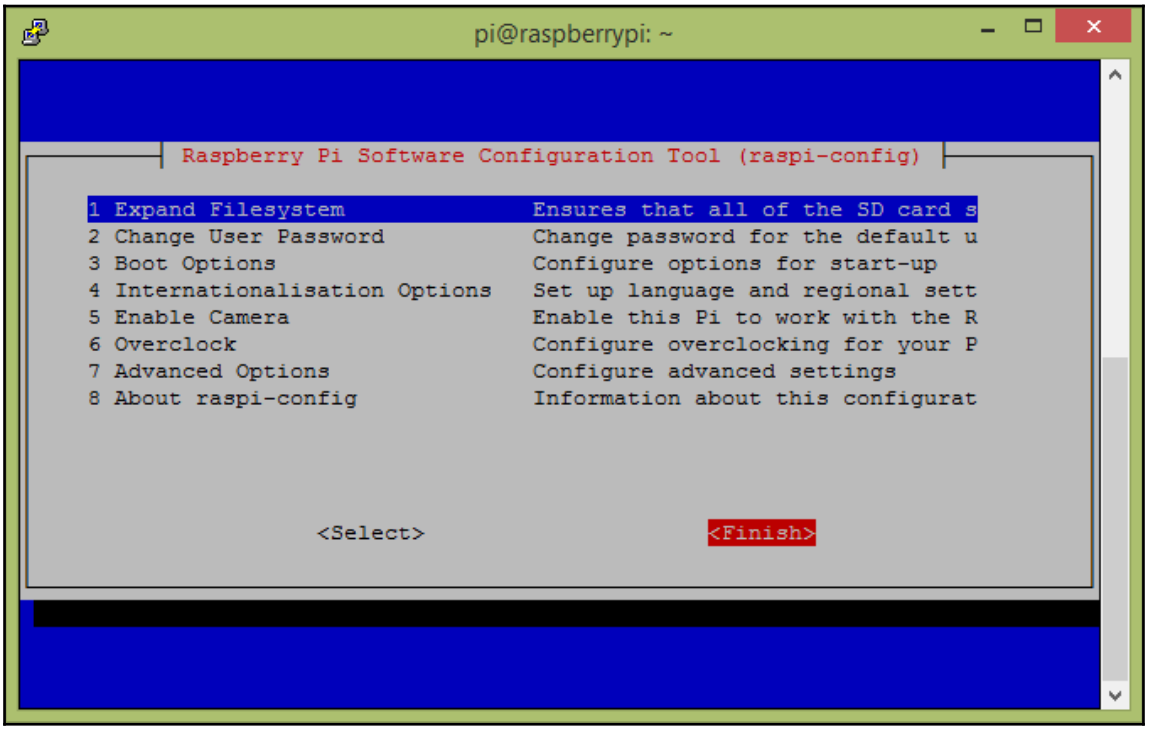


Figure 4-27: Configuration tool window

8. Finally, reboot the Raspberry Pi.

You have now successfully enabled the I2C.1 interface on the Raspberry Pi and it is ready to work with the I2C serial communication protocol and I2C-enabled devices.

Searching I2C devices attached to the Raspberry Pi

Before we start coding with `Pi4j`, it's time to search the `/dev/i2c-1` file system node for all addresses. The following command can be used to find the addresses of all the devices that are connected to the Raspberry Pi through the `I2C.1` interface:

```
sudo i2cdetect -y 1
```

You will get an output on the PuTTY terminal that looks similar to this:

```
     0 1 2 3 4 5 6 7 8 9 a b c d e f

00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

70: -- -- -- -- -- -- -- --
```

You still can't see any device addresses because you haven't connected any device to the I2C.1 interface yet.

Figure 4-28 shows the PuTTY terminal output for the `sudo i2cdetect -y 1` command; note that the temperature sensor attached to the Raspberry Pi is shown at the address, `0x48`:

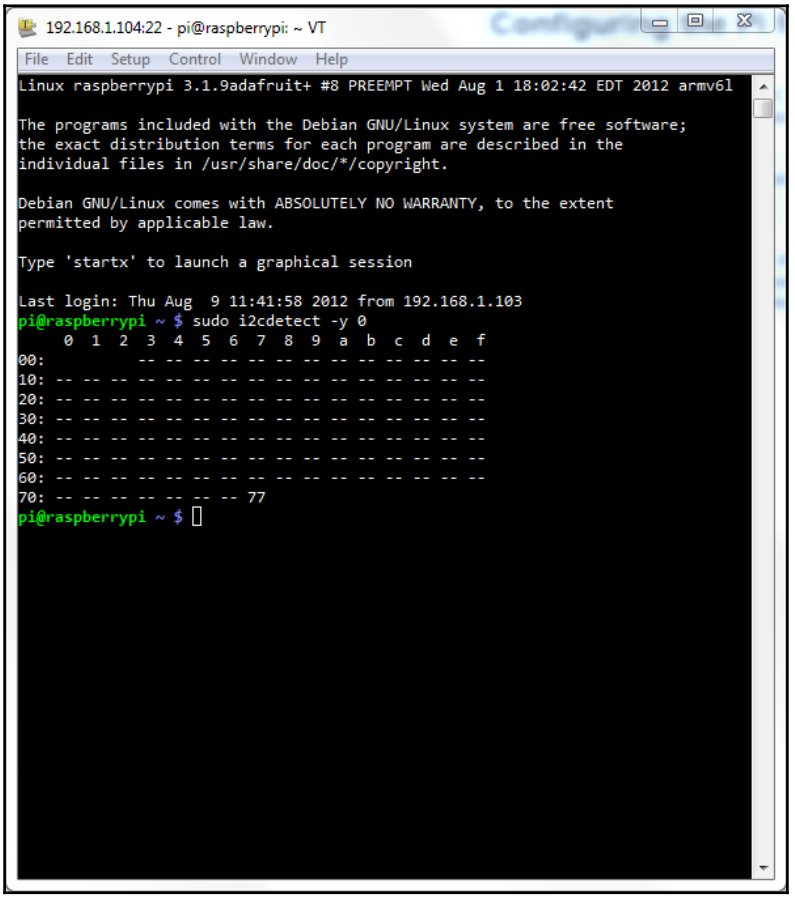


Figure 4-28: I2C address for the temperature sensor

This indicates that your TMP102 temperature sensor is correctly connected to I2C bus 1.

Accessing I2C with Pi4J

Pi4J provides a set of native methods for interacting with the I2C bus on the Raspberry Pi through the `I2C` class, which is a derived class of the `Object` class. The full reference for the `I2C` class can be found at `http://pi4j.com/apidocs/com/pi4j/jni/I2C.html`:

1. To add the `pi4j-core.jar` file, right-click on the **Libraries** folder and in the context menu, click **Add JAR/Folder.**
2. The **Add JAR/Folder** dialog box will appear to locate the jar file for the `pi4j` library (Figure 4-29). The `pi4j-core.jar` file is in the `pi4j-1.1\lib` folder. Select the `pi4j-core` file and click the **Open** button:

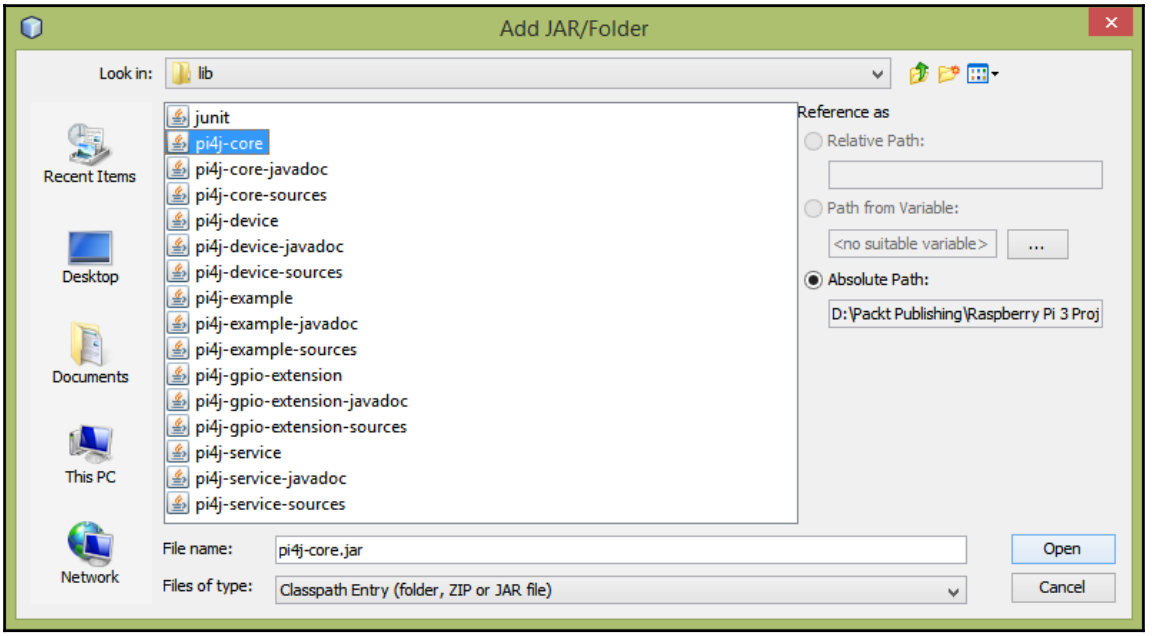


Figure 4-29: Locating the pi4j library

3. A new node, `pi4j-core.jar`, will be added under the `Libraries` node.

Eclipse Paho Java client

To publish and subscribe to a feed, you should have the MQTT client installed in your Raspberry Pi. A well-written MQTT client for working with Java is Eclipse Paho (`https://eclipse.org/paho/clients/java/`). The pre-built `Eclipse Paho client` library for Java can be downloaded from `https://repo.eclipse.org/content/repositories/paho-releases/org/eclipse/paho/mqtt-client/0.4.0/`.

1. Download and save the `mqtt-client-0.4.0.jar` file to your computer's local drive.
2. Open the Netbeans IDE and expand the **IoT Dashboard** project tree. Then, right-click on the `Libraries` folder and in the context menu, click **Add JAR/Folder**. Then, browse and select the downloaded `mqtt-client-0.4.0.jar` file and click **Open**. The `mqtt-client-0.4.0` library will be added to your project and ready to use with your Java code.

Writing Java program to publish data to a feed

Now, you're ready to learn how to write a simple Java program with the Eclipse Paho Java client to publish data to a feed. You have already set up the **Temperature** feed with Adafruit IO. Listing 4-1 presents the Java program that can be used to publish random values between 0-100 to the **Temperature** feed every 60 seconds.

Type all the lines from listing 4-1 into the `IoTDashboard.java` file. The `IoTDashboard.java` file includes the `main` method.

Listing 4-1: `IoTDashboard.java`

```
    package com.packt.B05688.chapter4;

import org.eclipse.paho.client.mqttv3.MqttClient;
import
    org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import
 org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.*;

/*
 *
 * @author Pradeeka
 */
```

```
public class IoTDashboard {

public static void main(String[] args) {
 try {
MqttClient client = new MqttClient(
 "tcp://io.adafruit.com:1883",
 MqttClient.generateClientId(),
 new MemoryPersistence());

 MqttConnectOptions options = new
     MqttConnectOptions();
 options.setUserName("pradeeka");
 options.setPassword("c62d5f647eb44a4098ff0c46403bd639".toCharArray());
 client.connect(options);

while (true) {
 try {
 client.publish("pradeeka/feeds/Temperature",
     Integer.toString((int)(Math.random() *
     101)).getBytes("UTF8"), 0, true);
 Thread.sleep(60*1000);
} catch (Exception e) {
 e.printStackTrace();
 }
//client.disconnect();
 }
 } catch (MqttException me) {
 me.printStackTrace();
 }
}
}
```

Let's examine the program to understand the most important steps:

1. Import all the required packages into the program:

```
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import
  org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.*;
```

2. Create an instance of `MqttClient` using the following parameters: URL, client identifier, and persistence. The URL can be constructed by combining host and port number. For Adafruit IO, the host is `io.adafruit.com` and the port number is 1883. Also, mention the protocol as tcp. The complete URL would be `tcp://io.adafruit.com:1883`. Make sure to avoid trailing backslashes(`/`), as it will result in a compilation error. The `generateClientId` method can be used to generate a temporary and unique client ID. As the final parameter, use in-memory persistence because we don't want the state to persist:

```
 MqttClient client = new MqttClient(
"tcp://io.adafruit.com:1883",
MqttClient.generateClientId(),
new MemoryPersistence());
```

3. The `MqttConnectOptions` class holds all the options related to the connection, such as username and password. The `username` is your Adafruit account username, and the `password` is your API key (not the password of your Adafruit account):

```
MqttConnectOptions options = new
MqttConnectOptions();
 options.setUserName("pradeeka");
options.setPassword("c62d5f647eb44a4098ff0c46403bd639".toCharArray(
));
```

4. Then, connect to Adafruit IO as a client using the `connect` method with `options`:

```
client.connect(options);
```

5. To publish data to the feed, use the `publish` method with the topic, value (payload), quality of service, and retained flag parameters. For simplicity, we've used the random method to generate integer values between 0-100 as the temperature data. The value of quality of service should be `0`, and set the retained flag as `true`. The `while` loop executes the `publish` method every 60 seconds with the use of `Thread.sleep(ms)`. With Adafruit IO, the current rate limit is at two requests per second.

The `disconnect` method is commented to avoid disconnecting the client from Adafruit IO:

```
while (true) {
 try {
 client.publish("pradeeka/feeds/Temperature",
Integer.toString((int)(Math.random() * 101)).getBytes("UTF8"), 0, true);
```

```
 Thread.sleep(60*1000);
} catch (Exception e) {
 e.printStackTrace();
 }
//client.disconnect();
 }
```

6. Now you're ready to test the Java program with Adafruit IO, using random values instead of the real sensor data. Press *F11*, or in the menu bar click **Run|Build Project (IoTDashboard)** to build the project. Correct the code if you get any errors. Then run the project by pressing *F6*, or in the menu bar click **Run|Run Project (IoTDashboard)**.

Now visit your Temperature feed page in Adafruit IO (`https://io.adafruit.com/username/feeds/temperature`) and wait about 10 minutes. It displays a real-time data feed from your Java program on a Line Chart (Figure 4-30) and simultaneously in a table view (Figure 4-31) with the value and time created. Both the Line Chart and table view will update every 60 seconds as they receive data from your Java program:

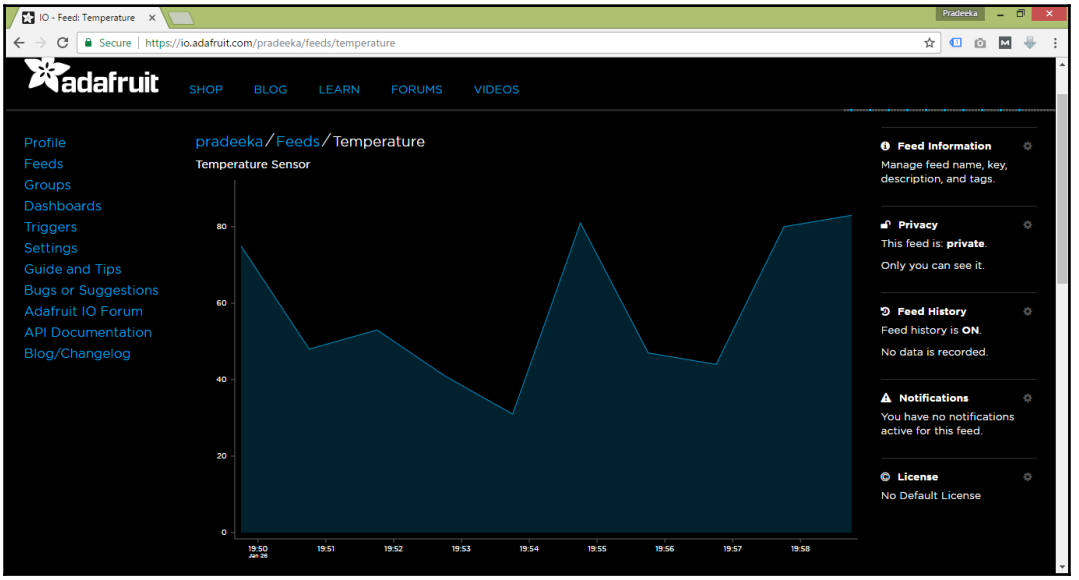


Figure 4-30: Displaying real-time data on a Line Chart

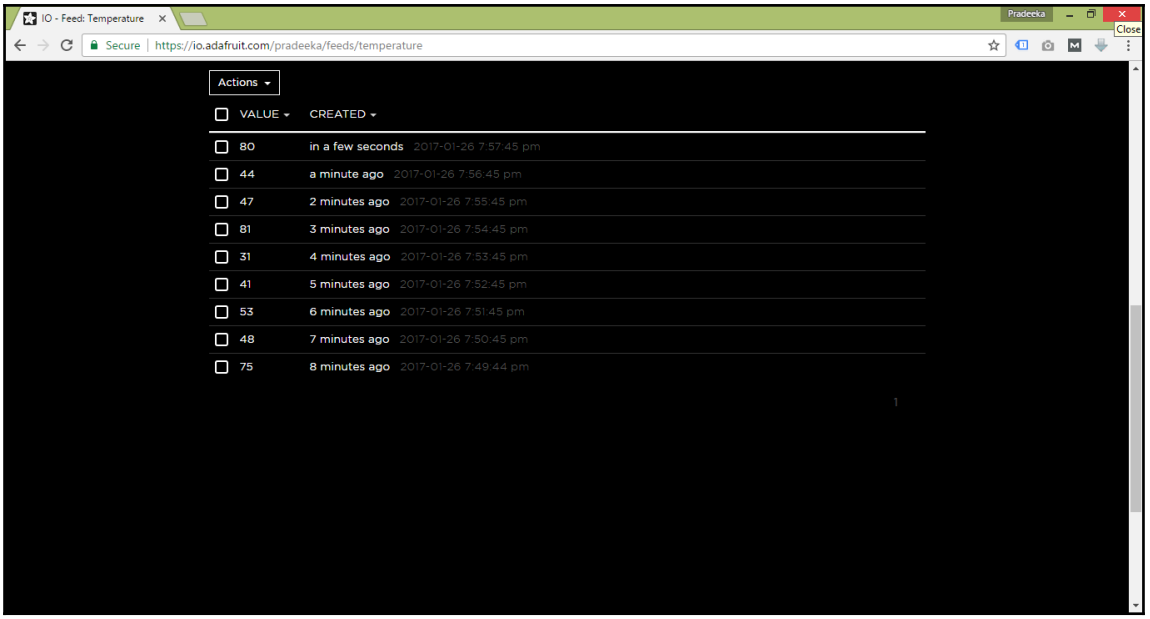


Figure 4-31: Real-time data in a table view

The Java program continuously sends data to the feed until you stop the program. You can stop the program by clicking on **Run | Stop Build/Run: IoTDashboard (run-remote)** from the menu bar.

Publishing temperature sensor data

Now we're going to implement simple code that can be used to read TMP102 temperature sensor data through the Raspberry Pi I2C bus 1. Listing 4-2 presents the Java class `TemperatureSensor.java`, which can be used to read the sensor data from TMP102 with the `pi4j` library. Add a new Java class file to your project by right-clicking on the **IoTDashboard** node, and from the context menu, click **New | Java Class**. Type `Temperature Sensor` in the Class `Name` text box and click the **Finish** button. An empty class file will be added to your project. Type all the lines listed under listing 4-2.

Listing 4-2: `TemperatureSensor.java`

```
    package com.packt.B05688.chapter4;

import com.pi4j.io.i2c.I2CBus;
import com.pi4j.io.i2c.I2CDevice;
import com.pi4j.io.i2c.I2CFactory;
import java.io.IOException;

/**
 *
 * @author Pradeeka
 */
public class TemperatureSensor {
private I2CBus bus = null;
private I2CDevice device = null;

public TemperatureSensor(int sensorAddress) {
 try {
 bus = I2CFactory.getInstance(I2CBus.BUS_1);
 if (bus != null) {
 device = bus.getDevice(sensorAddress);
 }
 } catch (Exception e) {
 System.out.println(e);
 }
 }

public double getTemperature() throws IOException {
 double result = getReadingFromDevice() * 0.0625;
 return result;
 }

 private int getReadingFromDevice() throws IOException {
int result = 0;

 if (device != null) {
 byte[] tempBuffer = new byte[2];
 int bytesRead = device.read(tempBuffer, 0, 2);
 if (bytesRead == 2) {
 int MSB = tempBuffer[0] < 0 ? 256 + tempBuffer[0] :
     tempBuffer[0];
 int LSB = tempBuffer[1] < 0 ? 256 + tempBuffer[1] :
     tempBuffer[1];
result = ((MSB << 8) | LSB) >> 4;
 }
 }
return result;
```

```
 }
}
```

Let's examine the important sections in the `TemperatureSensor` class:

1. First, import all the following packages to your Java project:

```
import com.pi4j.io.i2c.I2CBus;
import com.pi4j.io.i2c.I2CDevice;
import com.pi4j.io.i2c.I2CFactory;
import java.io.IOException;
```

2. Initialize the instances of `I2Cbus` and `I2CDevice`:

```
 private I2CBus bus = null;
 private I2CDevice device = null;
```

3. The `getInstance` method creates a new `I2CBus` instance and the `getDevice` method initializes the device with the particular address. The `sensorAddress` parameter holds the address of the TMP102 sensor, which is 0x48:

```
 public TemperatureSensor(int sensorAddress) {
 try {
 bus = I2CFactory.getInstance(I2CBus.BUS_1);
 if (bus != null) {
 device = bus.getDevice(sensorAddress);
 }
 } catch (Exception e) {
 System.out.println(e);
 }
 }
```

4. The TMP102 sensor holds the temperature in a 12-bit read-only register. The following method can be used to get the value stored in the temperature register:

```
private int getReadingFromDevice() throws
IOException {
 int result = 0;

 if (device != null) {
 byte[] tempBuffer = new byte[2];
 int bytesRead = device.read(tempBuffer, 0, 2);
 if (bytesRead == 2) {
 int MSB = tempBuffer[0] < 0 ? 256 + tempBuffer[0]
         :
         tempBuffer[0];
 int LSB = tempBuffer[1] < 0 ? 256 + tempBuffer[1]
```

```
            :
            tempBuffer[1];
result = ((MSB << 8) | LSB) >> 4;
 }
 }
return result;
 }
```

5. Simply multiply the result returned by `getReadingFromDevice` by `0.0625` to convert it to degrees Celsius:

```
public double getTemperature() throws IOException {
double result = getReadingFromDevice() * 0.0625;
return result;
}
```

6. A simple modification needs to be done in the `IoTDashboard.java` file to use the `TemperatureSensor` class.
7. Initialize the device address of the TMP102 temperature sensor, which is `0x48`:

```
private static final int DEVICE_ADDRESS = 0x48;
```

8. Create a new instance of the `TemperatureSensor` class inside the `main` method:

```
TemperatureSensor tmp102 = new
TemperatureSensor(DEVICE_ADDRESS);
```

9. Replace this line:

```
client.publish("pradeeka/feeds/Temperature",
Integer.toString((int)(Math.random() *
101)).getBytes("UTF8"), 0, true);
```

   With this:

```
client.publish("pradeeka/feeds/Temperature",
Double.toString((double)
(tmp102.getTemperature())).getBytes("UTF8"), 0,
true);
```

10. Build the project by clicking **Run|Build Project (IoTDashboard)** or pressing *F11*. If you get any errors, correct them before running the code and then rebuild the project.
11. Run the project by clicking **Run|Run Project (IoTDashboard)** or pressing *F6*.

12. Visit your Temperature feed page in Adafruit IO (`https://io.adafruit.com/username/feeds/temperature`). You will get a Line Chart similar to the one shown in Figure 4-32 and a *Table view* as shown in Figure 4-33:

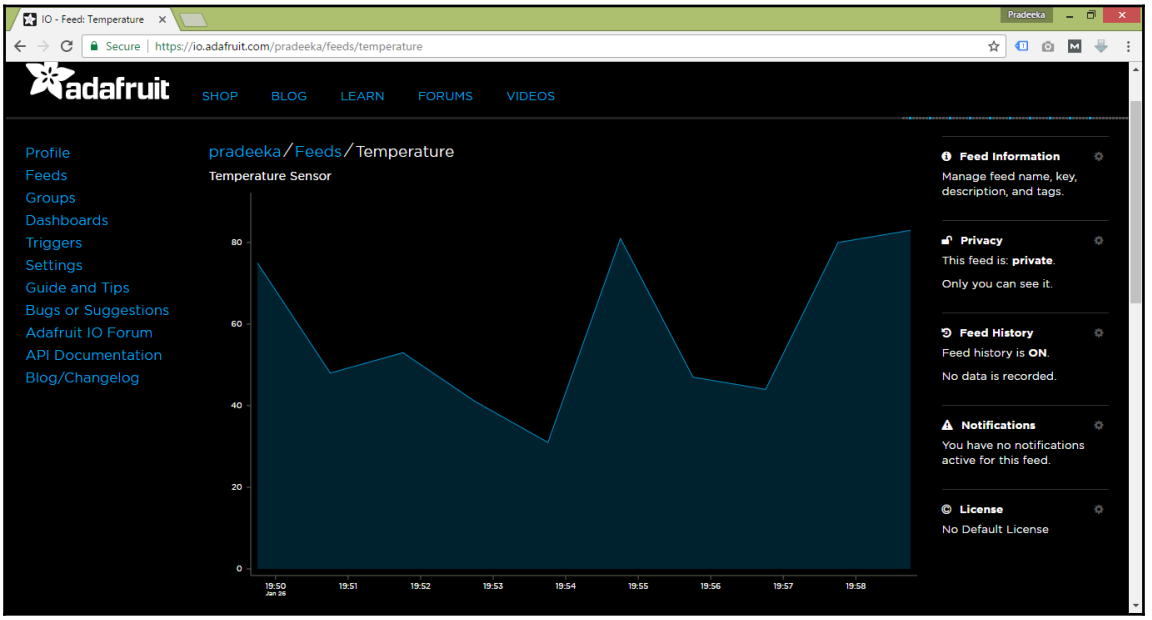


Figure 4-32: Display TMP102 temperature sensor data in Celsius on a Line Chart vs time

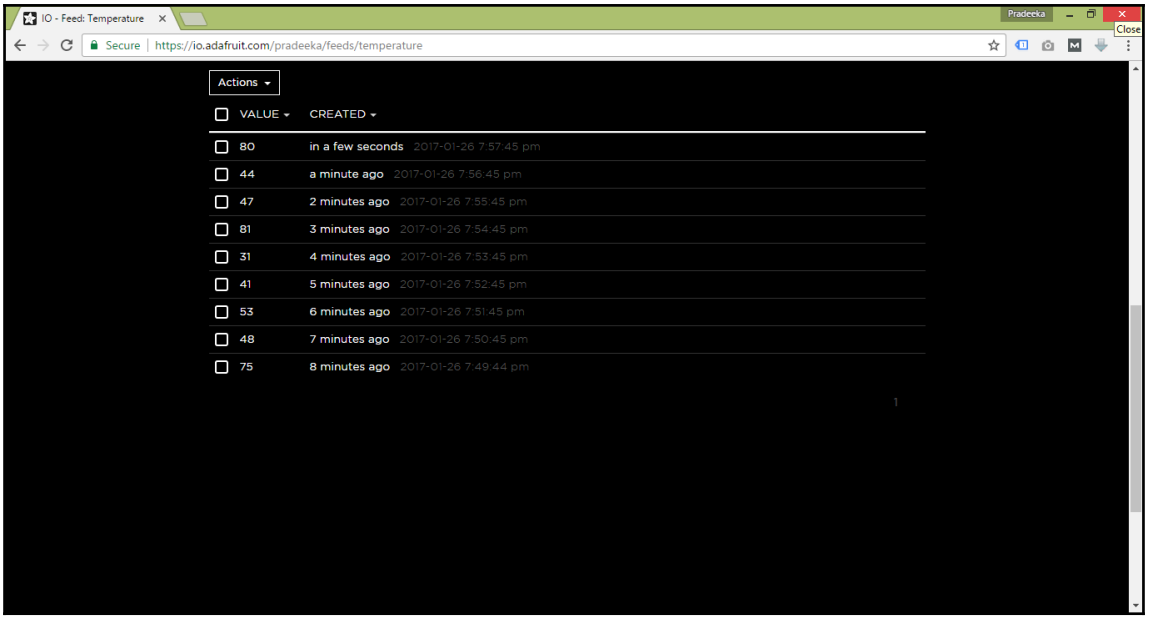


Figure 4-33: Real-time temperature data in table view

Publishing system information

Publishing Raspberry Pi system information to a topic is easy with the `pi4j` library. The `SystemInfo` class provides a host of methods to access Raspberry Pi system information. The Javadoc for the `SystemInfo` class can be found at `http://pi4j.com/apidocs/index.html?com/pi4j/system/SystemInfo.html`. The following steps will tell you how to publish the temperature of the CPU to the `username/feed/cpu-temperature` topic.

1. create a new feed named **CPU Temperature** before proceeding. Also, add a `Gauge` block to your IoT dashboard in the Adafriut IO system to display the CPU temperature.
2. To use the `SystemInfo` class with your program, first import the:

```
import com.pi4j.system.SystemInfo; package.
```

3. The `getCpuTemperature` method returns the current temperature of the CPU as a float value. The code for publishing the CPU temperature on the topic can be written as:

```
client.publish("pradeeka/feeds/cpu-
temperature",
Float.toString((float)(

SystemInfo.getCpuTemperature())).getBytes("UTF8"),
          0, true);
```

The Raspberry Pi CPU temperature will display on the IoT dashboard, as shown in Figure 4-34:

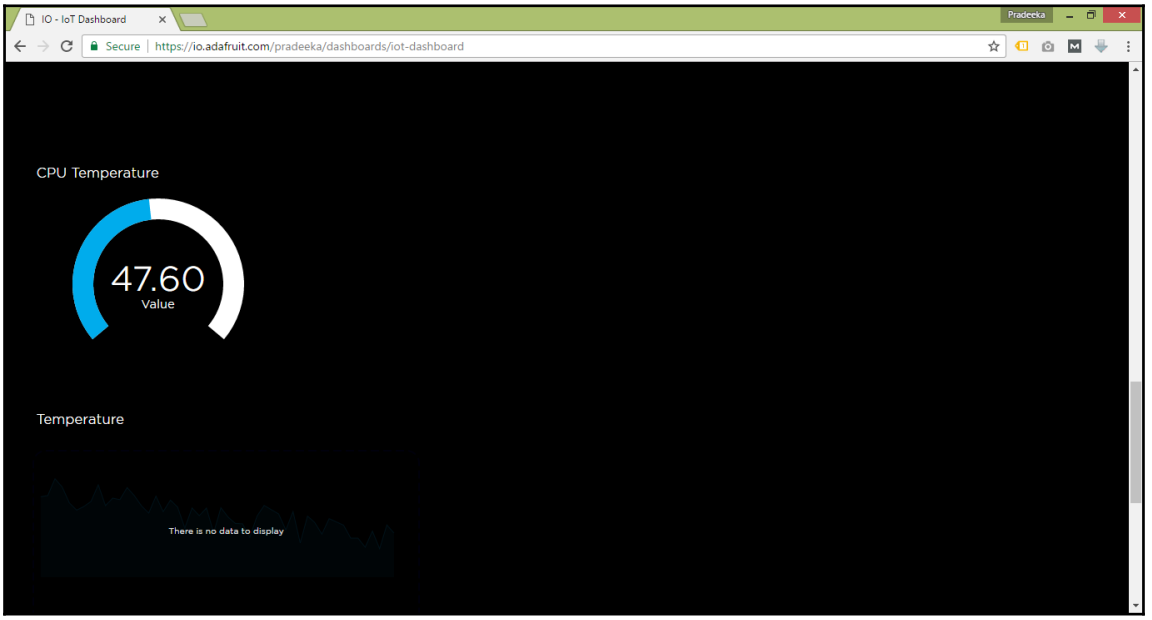


Figure 4-34: CPU temperature

Subscribing to a feed

You can subscribe to a feed to get data on a specific topic with Adafruit IO and the MQTT client. The feed can be generated by either a sensor or a UI control on the Adafruit dashboard.

Creating a toggle button on Adafruit dashboard

The Adafruit IO dashboard can be used to control any actuator attached to the Raspberry Pi. You can create the dashboard with various controllers such as **toggle** buttons, **momentary** buttons, **sliders**, **gauges**, **text**, **streams**, **images**, **Line Charts**, **color pickers**, and **maps**. The easiest way to create a user-driven feed is using a toggle button, which has only two states, on and off:

1. Create a new feed named **Button** under **Feeds** (Figure 4-35). The feed URL for the button will be similar to **username/feeds/** button:

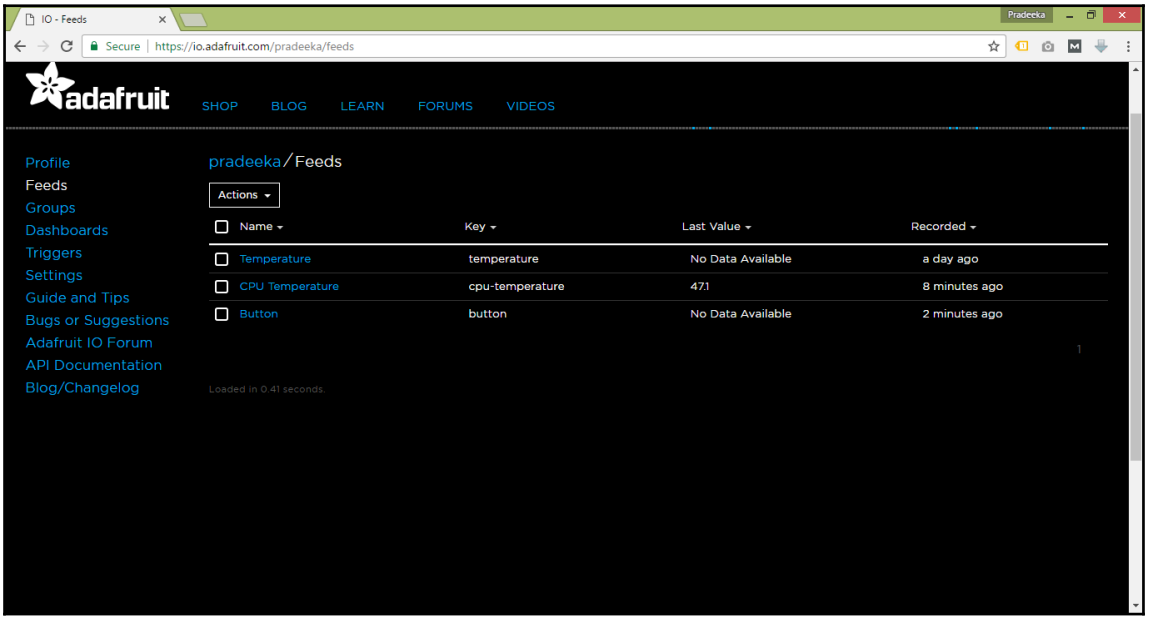


Figure 4-35: Feed for button

2. Open the IoT dashboard and click the **Create a new block** button in the top right-hand corner of the window.

3. Click the **Toggle** box (Figure 4-36):

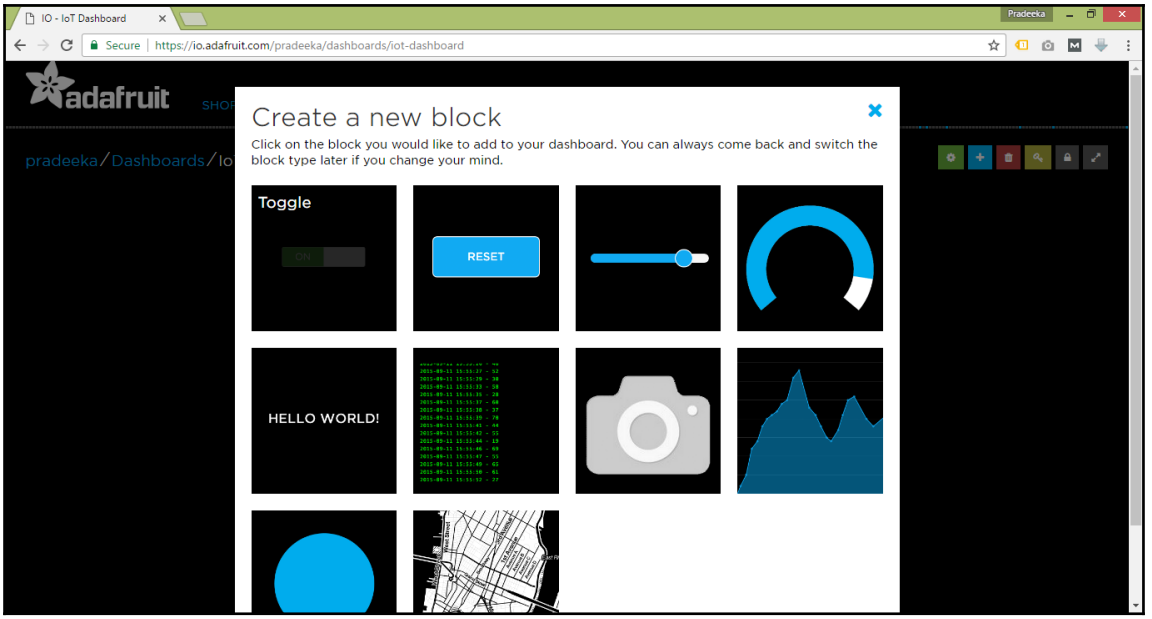


Figure 4-36: Toggle block

4. In the **Choose feed** dialog box, select the **Feed** button and click the **Next step>** button (Figure 4-37):

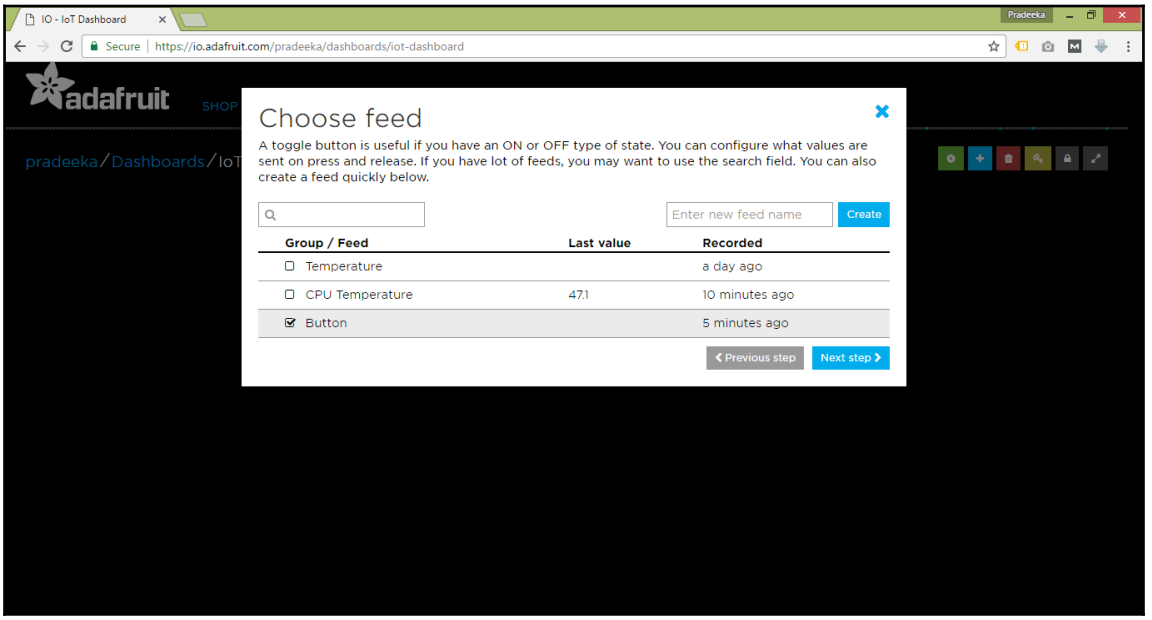


Figure 4-37: Assigning a feed to the block

5. In the **Block Settings** dialog box, click the **Create block** button to create the block (Figure 4-38). A new toggle button will be added to your dashboard in the default state, **OFF**:

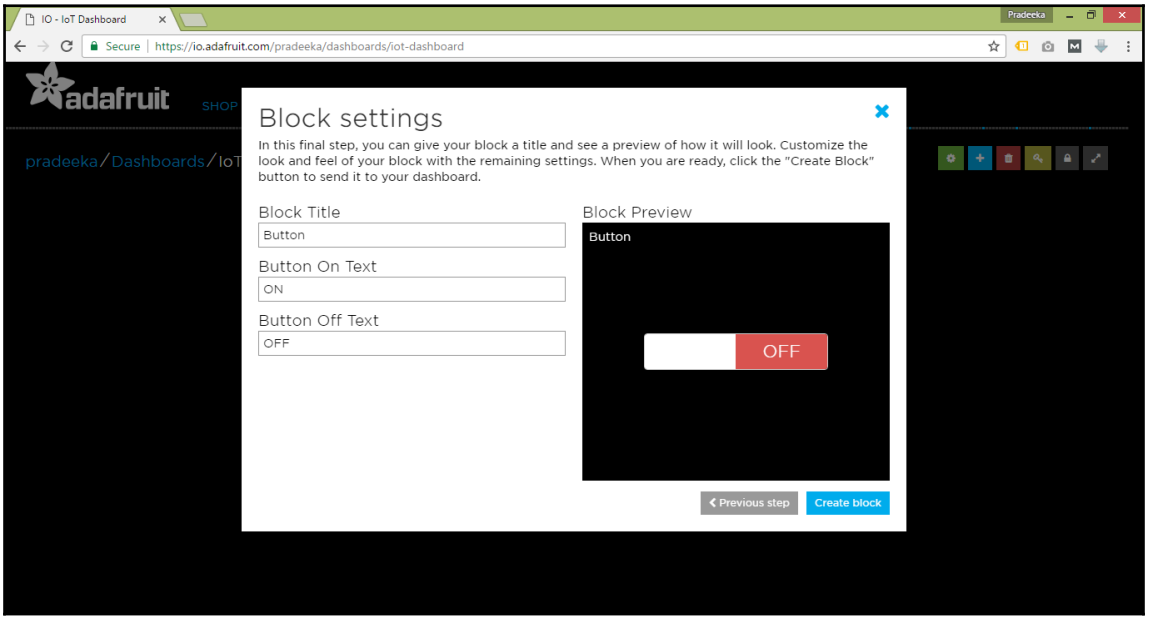


Figure 4-38: Settings for the button

Subscribe to the button feed

The following Java code snippets can be used to subscribe to the **username/feed/button** feed:

1. Open the `IoTDashboard.java` file and add the following two lines inside the `Import Packages` section:

```
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.
IMqttDeliveryToken;
```

2. Then, add the following method inside the `main` method:

```
client.setCallback(new MqttCallback() {

@Override
```

```
public void connectionLost(Throwable cause) {
}

@Override
public void messageArrived(String topic,
MqttMessage message)
throws Exception {
System.out.println(topic + ": " + message);
}

@Override
public void
deliveryComplete(IMqttDeliveryToken
token) {
        }
});
```

3. The subscribe method can be used to get the real-time data feeds generated by the toggle button. Set the quality of service level to `0`:

```
client.subscribe("pradeeka/feeds/button", 0);
```

4. Build and run the program. Open the IoT dashboard and resize the browser window to see both the toggle button and the output window of the NetBeans IDE. Now click on the toggle button to change its state. The output window will immediately show the current state of the button (**ON** or **OFF**), as shown in Figure 4-39:

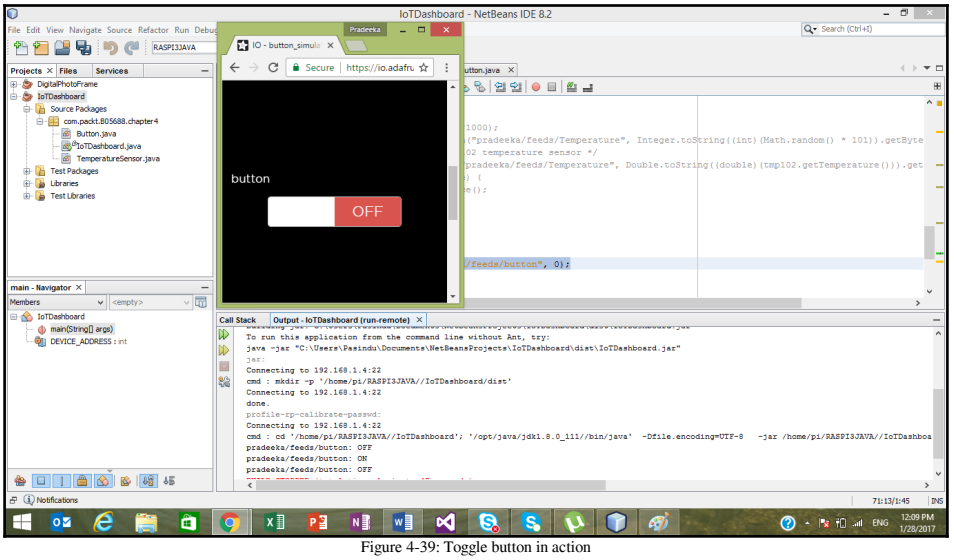


Figure 4-39: Toggle button in action

Controlling an LED from button feed

The text generated by the button can be used to control an LED attached to the Raspberry Pi. Figure 4-40 shows the connection diagram that you can use to build the circuit. You can do it with a few hookup wires, a 220-Ohm resistor, and a 3 mm LED:

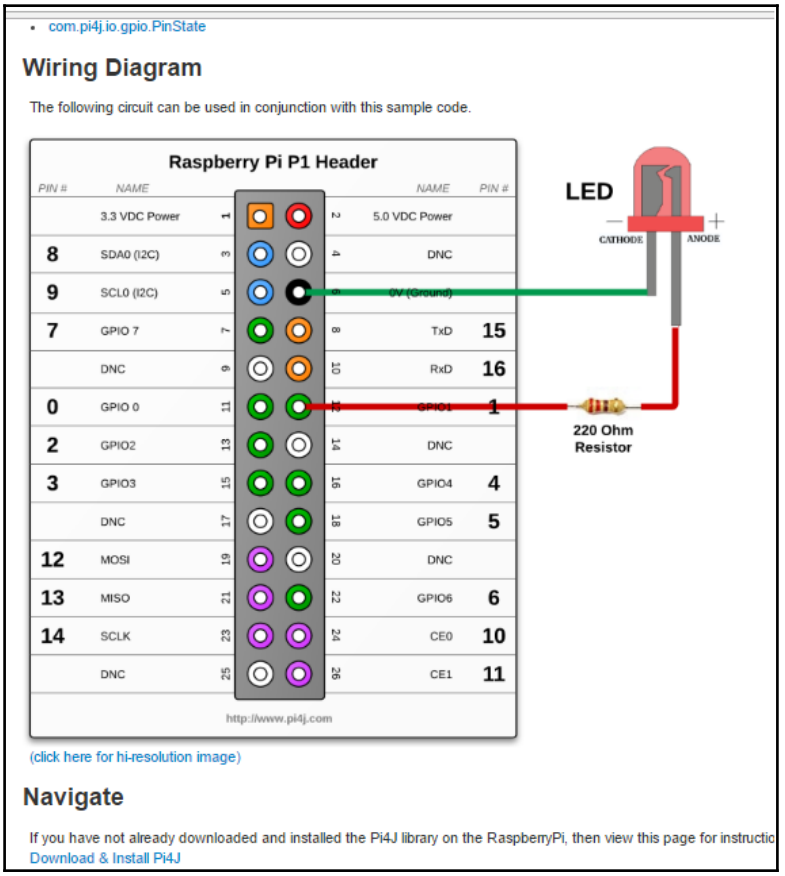


Figure 4-40: LED is connected to GPIO pin 1. Image courtesy of `http://www.pi4j.com`

1. Create a new Java class named `LED` in your project and type all the lines shown in listing 4-3:

Listing 4-3: `LED.java`

```
package com.packt.B05688.chapter4;

import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import
com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;

/**
*
* @author Pradeeka
*/
public class LED {

         final GpioController gpio;
         final GpioPinDigitalOutput pin;

         public LED()
         {
         gpio = GpioFactory.getInstance();
         pin =

  gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01,
         "MyLED", PinState.HIGH);
         pin.setShutdownOptions(true, PinState.LOW);
         }

         public void on()
         {
         pin.low();
         }

         public void off()
         {
         pin.low();
         }
        }
```

The `constructor` method first creates the GPIO controller, then provisions GPIO pin 1 as an output pin and turns it on, and finally sets the shutdown state for the pin. The two methods `on` and `off` can be used to turn the LED on and off:

1. Create an instance of the `LED` class in the `IoTDashboard.java` file:

```
LED led = new LED();
```

2. Add following code block inside the `messageArrived` method:

```
if(message.toString() == "OFF"){
led.off();
}
if(message.toString() == "ON"){
led.on();
}
```

3. Finally, build and run the project.
4. Now, open the IoT dashboard in your Adafruit IO account and click on the **Toggle** button. The state of the button will change to **ON**. The action will trigger the LED attached to GPIO pin 1 and turn it ON. Again, click on the **Toggle** button to turn the LED `OFF`. If your setup doesn't work, or if you experience any unexpected behavior with the LED, check the connections between the Raspberry Pi and the LED. Also, make sure you've attached the LED to the correct GPIO pin, which is GPIO pin 1.

Figure 4-41 shows the complete IoT dashboard with blocks for temperature and **CPU Temperature**, and a button for controlling the LED attached to the Raspberry Pi:

Figure 4-41: Complete IoT dashboard

Summary

In this chapter, you have learned how to build a sensor-based IoT dashboard with Adafruit IO and the Eclipse Paho MQTT Java client. You can improve this project by adding more sensors and actuators to your dashboard to build a rich IoT application. Also, you can use Adafruit IO to control and monitor your home appliances remotely with a handheld device such as a mobile phone, iPad, or iPod.

`Chapter 5`, *Wireless Controlled Robot*, is based on robotics, and you'll build a wirelessly-controlled robot with a Pololu Zumo chassis robot kit. Have fun!

5
Wireless Controlled Robot

Raspberry Pi-based robotic applications have become more popular because it is more than powerful enough to control a robot with a range of embedded operating systems. The Raspberry Pi is physically small in size and allows you to build more compact robots by attaching various sensors and actuators. The behavior of a robot can be simple (move forward or turn left) to complex (line following or solving a maze) depending on the problem it has to solve.

Pololu (`https://www.pololu.com`) provides series of do-it-yourself robotic chassis kits such as Romi, Zumo, 3pi, and Tamiya. In this chapter, we will be choosing the Zumo chassis kit to build our Raspberry Pi-based wirelessly-controlled Zumo robot because it is cheap, easy to assemble, and easy to use.

In this chapter, you will:

- Learn how to assemble the Zumo chassis kit with motors (external reference)
- Assemble the Raspberry Pi 3 and motor driver with the Zumo chassis
- Drive micro metal gear motors with the H-bridge motor driver - SN754410 breakout board
- Reduce the electrical noise of the gear motors by using capacitors
- Use WiringPi to make **Pulse Width Modulation** (**PWM**) on Raspberry Pi GPIO pins with the `Pi4j` library
- Write a Java application to control the Zumo robot using PuTTY over a Wi-Fi network

A well known robotic application called the *line following robot* can be built with the Zumo chassis kit and Simulink. A very good tutorial on this can be found at `https://learn.adafruit.com/line-following-zumo-robot-programmed-with-simulink/overview`.

Prerequisites

Let's get started by preparing our tool box with the following hardware:

- One Zumo chassis kit (no motors), about US $19.95 at `https://www.pololu.com/product/1418`
- Two 100:1 micro metal gear motors HP 6V, about US $15.95 each at `https://www.pololu.com/product/1101`
- One H-bridge motor driver - SN754410 breakout board, about US $34.95 at `http://pcbgadgets.com/index.php?route=product/product&product_id=64`
- Aluminum standoff: 1/4" length, 2-56 thread, M-F (four-pack), about US $1.29 at `https://www.pololu.com/product/1940`
- Aluminum standoff: 3/4" length, 2-56 thread, M-F (four-pack) about US $1.39 at `https://www.pololu.com/product/1943`
- Machine screw: M3, 5 mm length, Phillips (25-pack) about US $0.99 at `https://www.pololu.com/product/1075`
- Two 0.1µF ceramic capacitors
- Four rechargeable NiMH AA batteries (each 1.2V, 2200 mAh)
- One USB battery pack for the Raspberry Pi, 10000 mAh with 2 x 5V outputs, about US $39.95 at `https://www.adafruit.com/product/1566`

The Zumo chassis kit

The **Pololu Zumo chassis** is a small, tracked robot platform, less than 10 cm on each side. It allows you to build custom robotic applications by connecting a variety of Micro Metal Gear motors to get a combination of torque and speed. The Zumo chassis can be purchased as a kit to build our Raspberry Pi wireless controlled robot.

The Zumo Chassis Kit contains the following mechanical components, shown in Figure 5-1:

- Zumo chassis main body
- 1/16″ black acrylic mounting plate
- Two drive sprockets
- Two idler sprockets
- Two 22-tooth silicone tracks
- Two shoulder bolts with washers and M3 nuts
- Four 1/4″ #2-56 screws and nuts

- Battery terminals

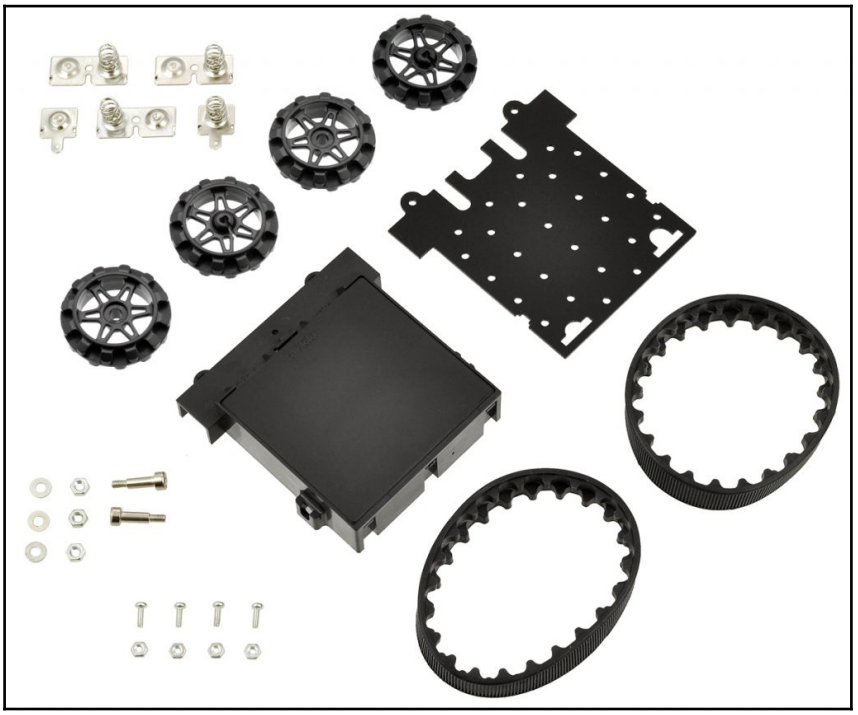


Figure 5-1: Zumo chassis kit components. Image credits: Courtesy of Pololu (`https://www.pololu.com`)

Assembling Zumo chassis

Assembling the Zumo chassis is easy with the following tools:

- Small Phillips screwdriver
- 3 mm Allen wrench (hex key)

The complete assembly instructions for the Zumo chassis can be found at `https://www.pololu.com/docs/0J54`.

Preparing motors to reducing the effects of electrical noise

After assembling the Zumo chassis, prepare the two motors to reduce the effects of electrical noise. Motors produce a large amount of electrical noise when they are operating. Using a bypass capacitor across the motor terminals is one of the solutions to reduce the electrical noise.

You can use a single ceramic capacitor across the motor terminals to reduce the electrical noise.

Solder a non-polarized **0.1 µF** ceramic capacitor across the motor terminals, as shown in Figure 5-2. These capacitors are very cheap and easily found at any electronics components shop:

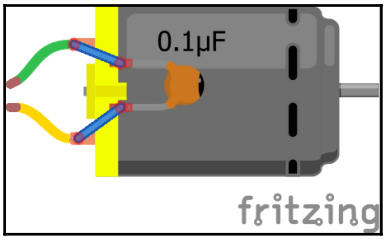


Figure 5-2: 0.1 µF ceramic capacitor soldered across the motor terminals

If you need more electrical noise reduction, use one of the following methods:

1. Solder two capacitors to your motor, one from each motor terminal, to the motor case, as shown in figure 5-3:

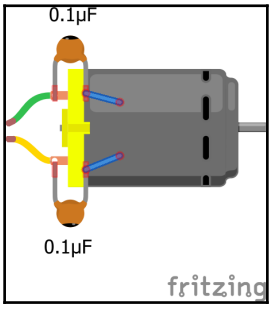


Figure 5-3: Two 0.1 µF ceramic capacitors soldered to the motor for greater noise reduction

2. Solder three capacitors to your motor, one across the terminals and one from each terminal to the motor case, as shown in Figure 5-4:

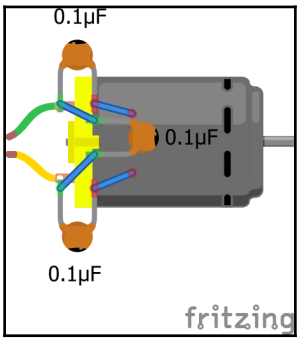


Figure 5-4: Three 0.1 µF ceramic capacitors soldered to the motor for greater noise reduction

You can also reduce the electrical noise by keeping the power leads of the motor as short as possible, twisting the power leads of the motor, keeping the motor and power wires away from the signal lines, and adding bypass capacitors across the power and ground lines.

Attaching Raspberry Pi to Zumo chassis

The 1/16" black acrylic mounting plate can be used to mount the Raspberry Pi with aluminum standoffs. First, attach four aluminum standoffs to the acrylic mounting plate. Figure 5-5 shows the section of the acrylic mounting plate with an aluminum standoff attached. Then, place the Raspberry Pi on top of the standoffs and secure it by using screws:

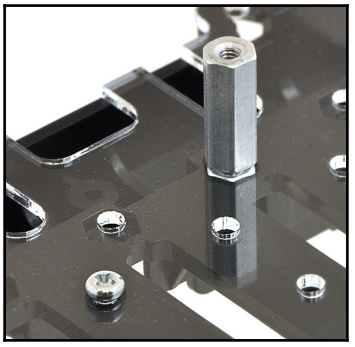


Figure 5-5: Connecting standoffs to the acrylic mounting plate. Image credits: Courtesy of Pololu (`https://www.pololu.com`)

Building the circuit

The circuit can be built with a SN754410 H-Bridge Motor Driver IC, as shown in Figure 5-6. The driver IC allows you to control four motors in one direction by using the four half H-Bridges, or control two motors in both directions by using a full H-Bridge for each motor. The Zumo robot requires two motors to drive the wheels in both directions using a full H-Bridge for each motor. Use four rechargeable NiMH AA-sized batteries with the Zumo chassis's battery compartment to power the motors and driver circuit:

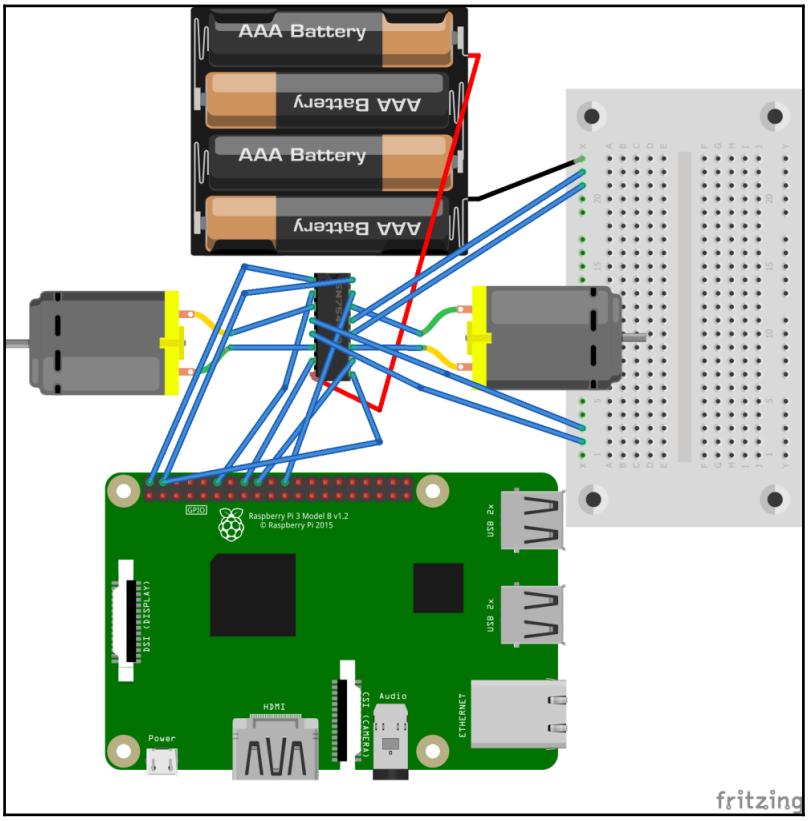


Figure 5-6: Motor driver connected to the Raspberry Pi

The pin layout of SN754410 IC is shown in Figure 5-7. The IC comes with a 16-pin NE package and it can operate in a wide voltage range of 4.5V to 36V DC. The output current capability per driver is 1 A. It also provides thermal shutdown when required:

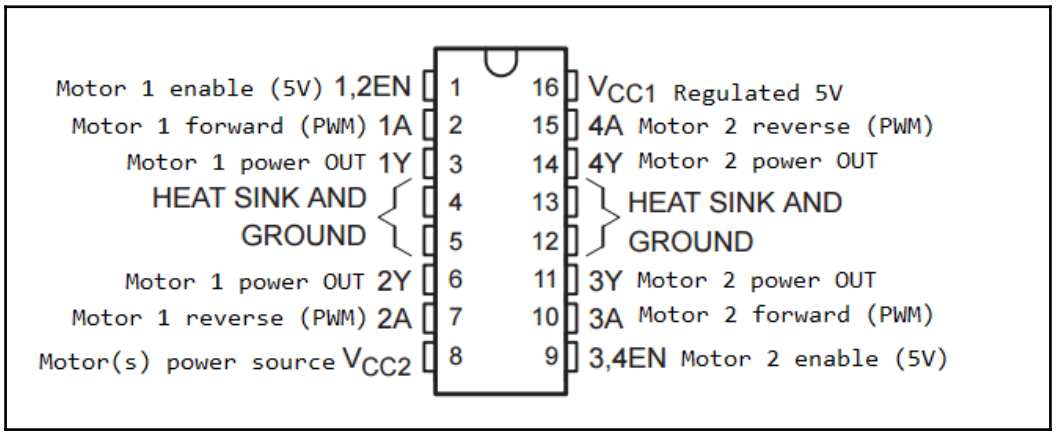


Figure 5-7: SN754410 pin layout

Table 5-1 presents the functions and specifications for each pin of the SN754410 IC:

Pin	Name	H-Bridge function	Specification
1	1,2EN	Motor 1 enable signal	Set 0V (LOW) on this pin to disable motor 1 or 5V (HIGH) to enable motor 1. Normally connect this pin to 5V.
2	1A	Motor 1 forward	Provide 5V to this pin to drive motor 1 at full speed in one direction. The motor speed can be controlled by PWM on the particular pin by changing the voltage between 0-5V.
3	1Y	Motor 1 power	Connect this pin to one lead of motor 1.
4	HEAT SINK AND GROUND	Ground	Connect this pin to Ground with a heat sink.
5	HEAT SINK AND GROUND	Ground	Connect this pin to Ground with a heat sink.
6	2Y	Motor 1 power	Connect this pin with the other lead of motor 1.

7	2A	Motor 1 reverse	Provide 5v to this pin to drive motor 1 at full speed in the opposite direction. The motor speed can be controlled with PWM on the particular pin by changing the voltage between 0-5V.
8	VCC2	Motor power source	Connect this pin to the power source (battery pack) for the motor. Maximum voltage is 36V.
9	3,4EN	Motor 2 enable	Set 0V (LOW) on this pin to disable motor 2 or 5V (HIGH) to enable motor 2. Normally connect this pin to 5V.
10	3A	Motor 2 forward	Provide 5V to this pin to drive motor 2 at full speed in one direction. The motor speed can be controlled by PWM on the particular pin by changing the voltage between 0-5V.
11	3Y	Motor 2 power	Connect this pin to the other lead of motor 2.
12	HEAT SINK AND GROUND	Ground	Connect this pin to Ground with a heat sink.
13	HEAT SINK AND GROUND	Ground	Connect this pin to Ground with a heat sink.
14	4Y	Motor 2 power	Connect this pin to the other lead of motor 2.
15	4A	Motor 2 reverse	Provide 5V to this pin to drive motor 2 at full speed in the opposite direction. The motor speed can be controlled by PWM on the particular pin by changing the voltage between 0-5V.
16	VCC1	IC logic power	Connect this pin to a regulated 5V source.

Table 5-1: Functions and specifications of the SN754410 IC pins

The battery compartment of the Zumo chassis can provide 4.6V with 4 x AA NiMH rechargeable batteries (Figure 5-8). A single battery can provide 1.2V with a capacity of 2200 mAh or more:

Figure 5-8: AA NiMH rechargeable battery - 1.2V, 2200mAh

The motor driver circuit is built on the SN754410 breakout board as shown in Figure 5-9. The fully assembled board comes with a removable SN754410 IC and a heat sink. The terminal blocks allow you to connect the board to motors, the Raspberry Pi, and power sources:

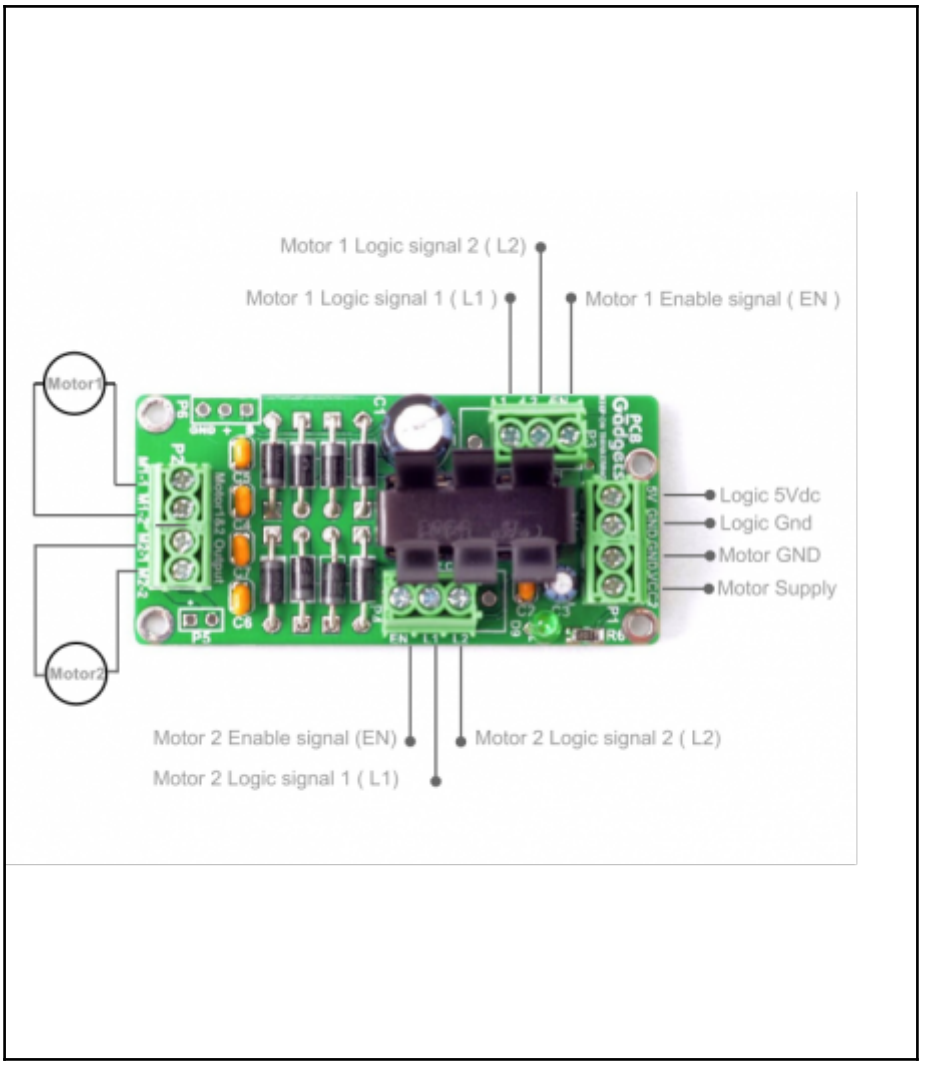


Figure 5-9: H-Bridge motor driver - SN754410 breakout board (top view). Image credits: Courtesy of PCB Gadgets (`http://pcbgadgets.com/`)

Wiring them together

The instructions given here explain how to connect the SN754410 breakout board to two motors and the Raspberry Pi, attached to the Zumo chassis.

Connect two motors to the SN754410 breakout board:

1. Connect the *l*eft side motor to the M1-1 and M1-2 terminals of terminal block P2.
2. Connect the right side motor to the M2-1 and M2-2 terminals of terminal block P2.

Connect Raspberry Pi GPIO to the SN754410 breakout board:

1. Connect the L1 terminal of terminal block P3 to GPIO pin 18 of the Raspberry Pi. This will drive the left side motor in a forward direction by enabling PWM on GPIO pin 18, which is equivalent to the WiringPi pin 1.
2. Connect the L2 terminal of terminal block P3 to GPIO pin 23 of the Raspberry Pi. This will drive the left side motor in a *backward* direction by enabling PWM on GPIO pin 23, which is equivalent to the WiringPi pin 4.
3. Connect the L1 terminal of terminal block P4 to GPIO pin 24 of the Raspberry Pi. This will drive the right side motor in a forward direction by enabling PWM on GPIO pin 24, which is equivalent to the WiringPi pin 5.
4. Connect the L2 terminal of terminal block P4 to GPIO pin 25 of the Raspberry Pi. This will drive the right side motor in a backward direction by enabling PWM on GPIO pin 25, which is equivalent to the WiringPi pin 6.

Connect power lines to the SN754410 breakout board:

1. With terminal block P1:
    1. Connect the 5V terminal to the Raspberry Pi pin 2 (in the J8 header).
    2. Connect the GND terminal to the Raspberry Pi pin 6 (in the J8 header).
    3. Connect the GND terminal to the battery compartment's NEGATIVE terminal in the Zumo chassis.
    4. Connect the VCC2 terminal to the battery compartment's POSITIVE terminal in the Zumo chassis.

2. With terminal blocks P3 and P4:
    - Connect the EN terminals to Raspberry Pi pin 2 or 4 (in the J8 header).

Don't insert batteries into the Zumo chassis battery compartment yet and just power your Raspberry Pi board using an external power supply, such as a wall wart.
For your reference, the internal connection between the SN754410 IC and the terminals of the breakout board is shown in table 5-2.

SN754410 IC pin	Terminal block	Terminal
3	P2	M1 -1
6	P2	M1 -2
11	P2	M2 -1
14	P2	M2 -2
2	P3	L1
7	P3	L2
10	P4	L1
15	P4	L2
1, 8, 16	P1	5V
4, 5, 12, 13	P1	GND
1, 9	P3	EN

Table 5-2: SN754410 IC pin and terminal mapping

Moving and turning

The moving and turning operations of the Zumo robot can be performed by controlling the rotation and rotating direction of the two wheels attached to the motors.

Moving

You can **move** the Zumo robot by controlling the rotation direction of the two wheels at the same time. The pseudo code can be written for forward (Figure 5-10) and backward (Figure 5-11) movements as follows:

Forward:

- Left motor forward
- Right motor forward

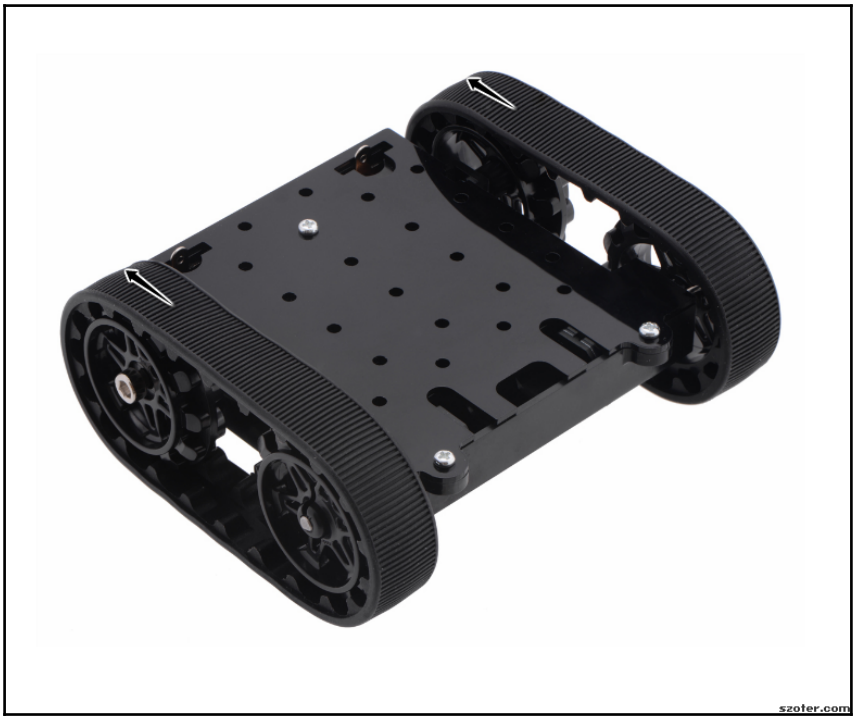


Figure 5-10: Moving forward. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

Backward:

- Left motor backward
- Right motor backward

Figure 5-11: Moving backward. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

Turning

You can turn the Zumo robot by using either the point turn or swing turn technique.

Point turn: A point turn requires that the center of rotation is centered between the drive wheels, as shown in Figure 5-12. This type of turning can be achieved by driving one motor forward while driving the other motor backward. The motor driving backward determines the direction of turn:

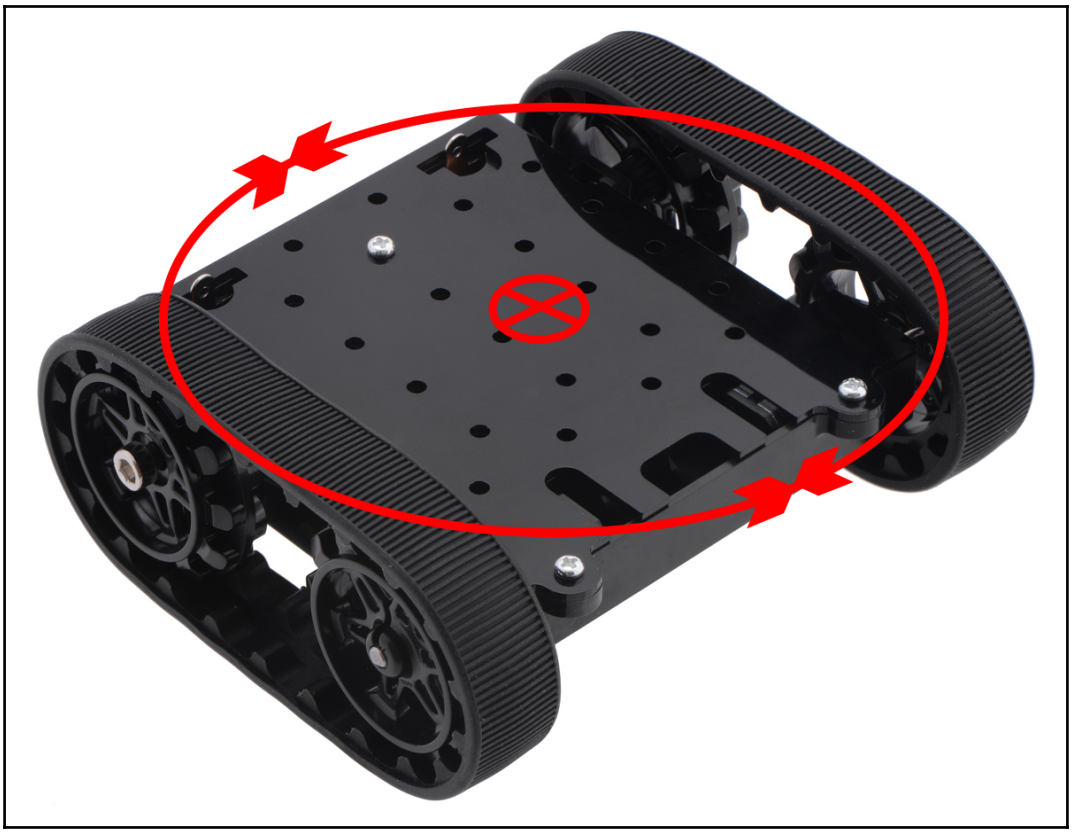


Figure 5-12: Turning paths for point turn. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

The pseudo code can be written for Point Turn Left (Figure 5-13) and Point Turn Right (Figure 5-14) as follows.

Point turn left:

- Left motor backwards
- Right motor forward

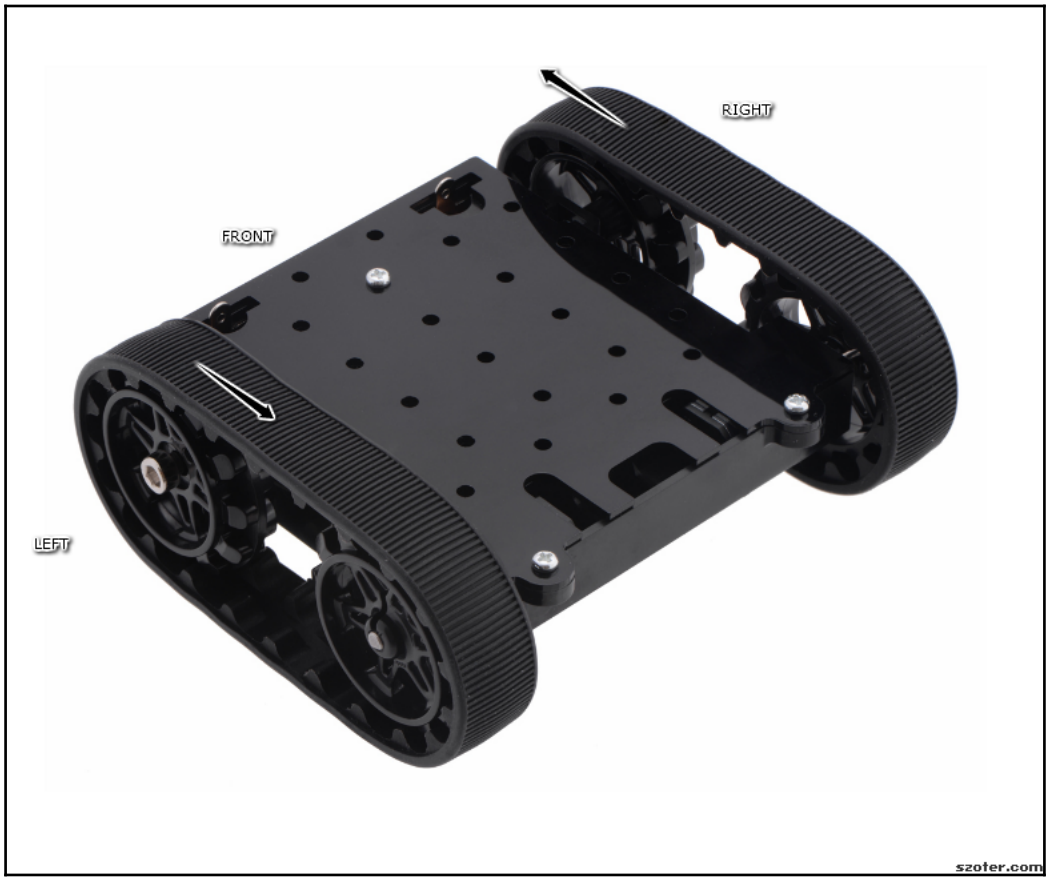


Figure 5-13: Point turn left. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

Point turn right:

- Right motor backwards
- Left motor forward

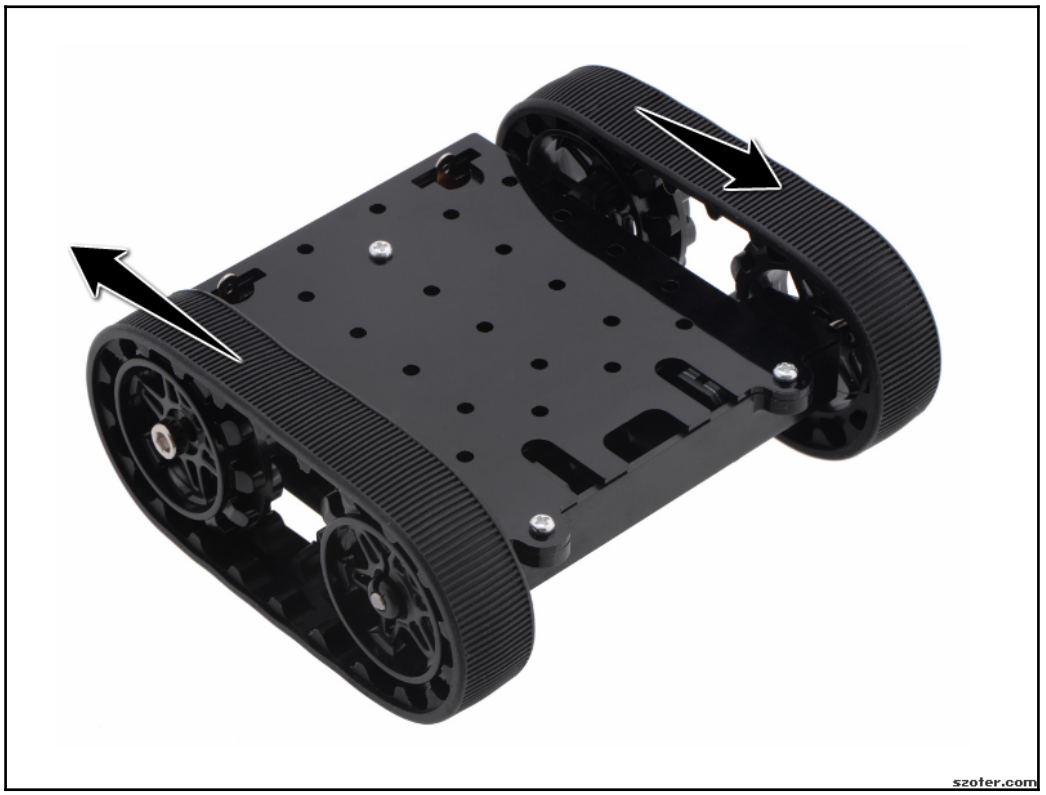


Figure 5-14: Point turn right. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

Swing turn

A swing turn is where a vehicle swings around a pivot point, as shown in Figure 5-15. A swing turn can be achieved by driving one motor forward or backward, while turning off the other motor. The turn type can be either forward swing turn left, backward swing turn left, forward swing turn right, or backward swing turn right:

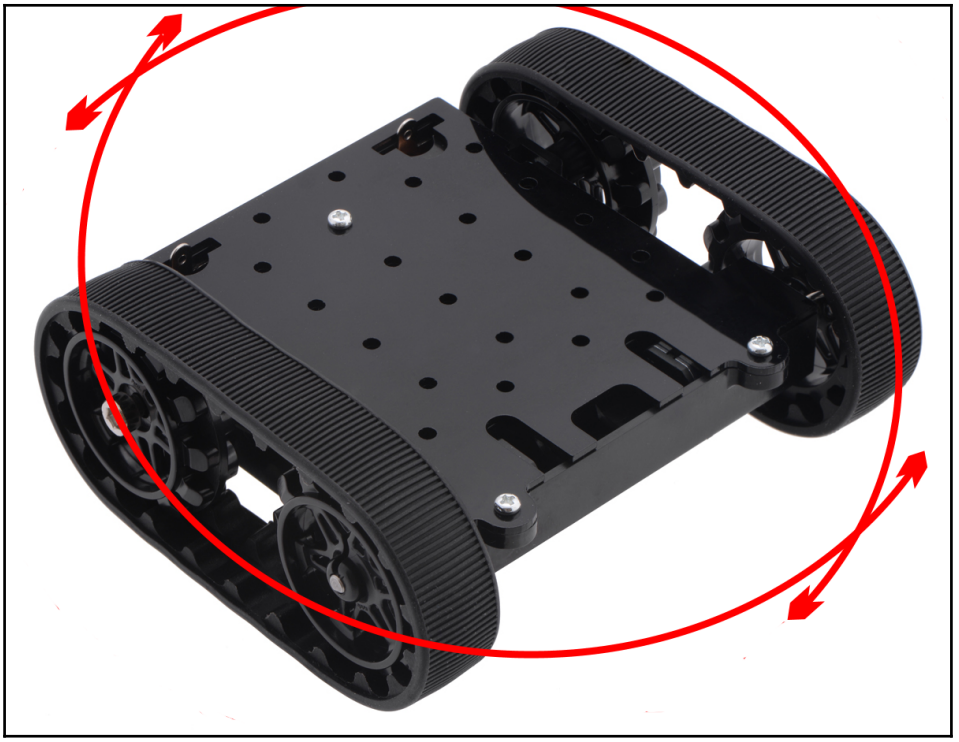


Figure 5-15: Turning paths for swing turn. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

The pseudo code can be written for forward swing turn left (Figure 5-16), backward swing turn left (Figure 5-17) , forward swing turn right (Figure 5-18), and backward swing turn right (Figure 5-19) as follows.

Forward swing turn left:

- Left motor off
- Right motor forward

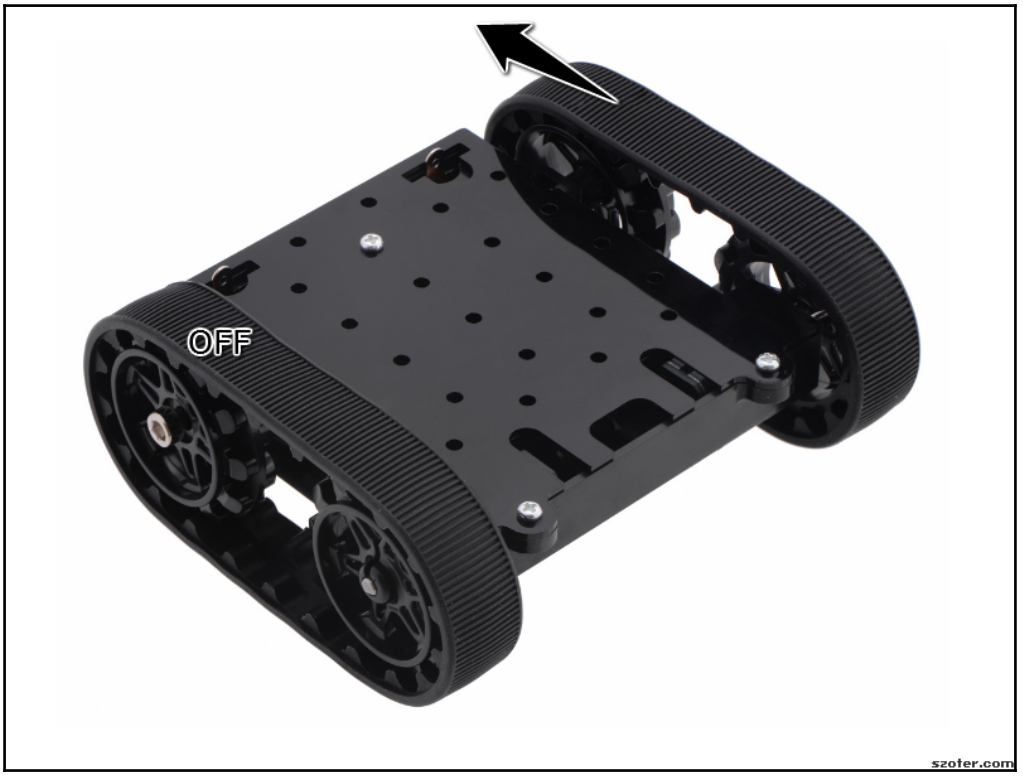


Figure 5-16: Forward swing turn left. Image credits: Courtesy of Plololu (`https://www.pololu.com`). Image edited for demonstration the action

Backward swing turn left:

- Left motor off
- Right motor backward

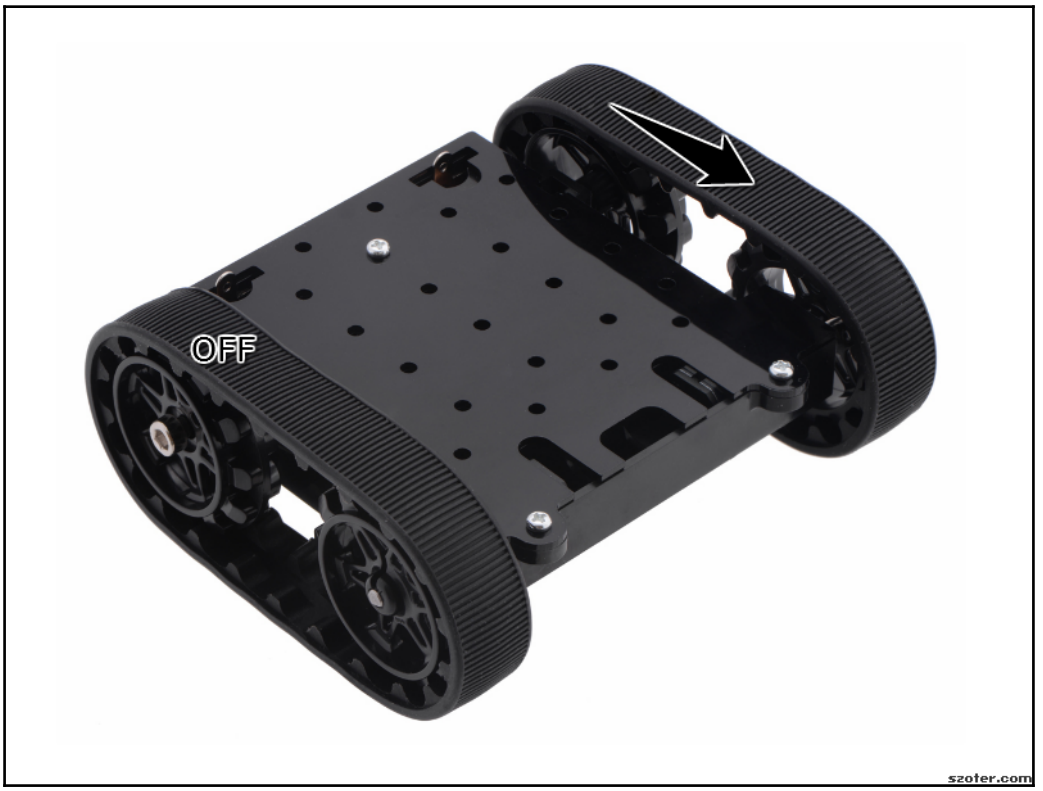


Figure 5-17: Backward swing turn left. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

Forward swing turn right:

- Right motor off
- Left motor forward

Figure 5-18: Forward swing turn right. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

Backward swing turn right:

- Right motor off
- Left motor backward

Figure 5-19: Backward swing turn right. Image credits: Courtesy of Pololu (`https://www.pololu.com`). Image edited for demonstration the action

You can use combinations of all or a few turning mechanisms to turn your Zumo robot, depending on your application requirements.

Writing your Java program

Open the NetBeans IDE and create a new project named `ZumoRobot`.

1. In the menu bar, click **File**|**New Project**. The **New Project** wizard will appear. In the **New Project** wizard, click on **Java** under **Categories** and click on **Java Application** under **Projects**. Then click the **Next** button:

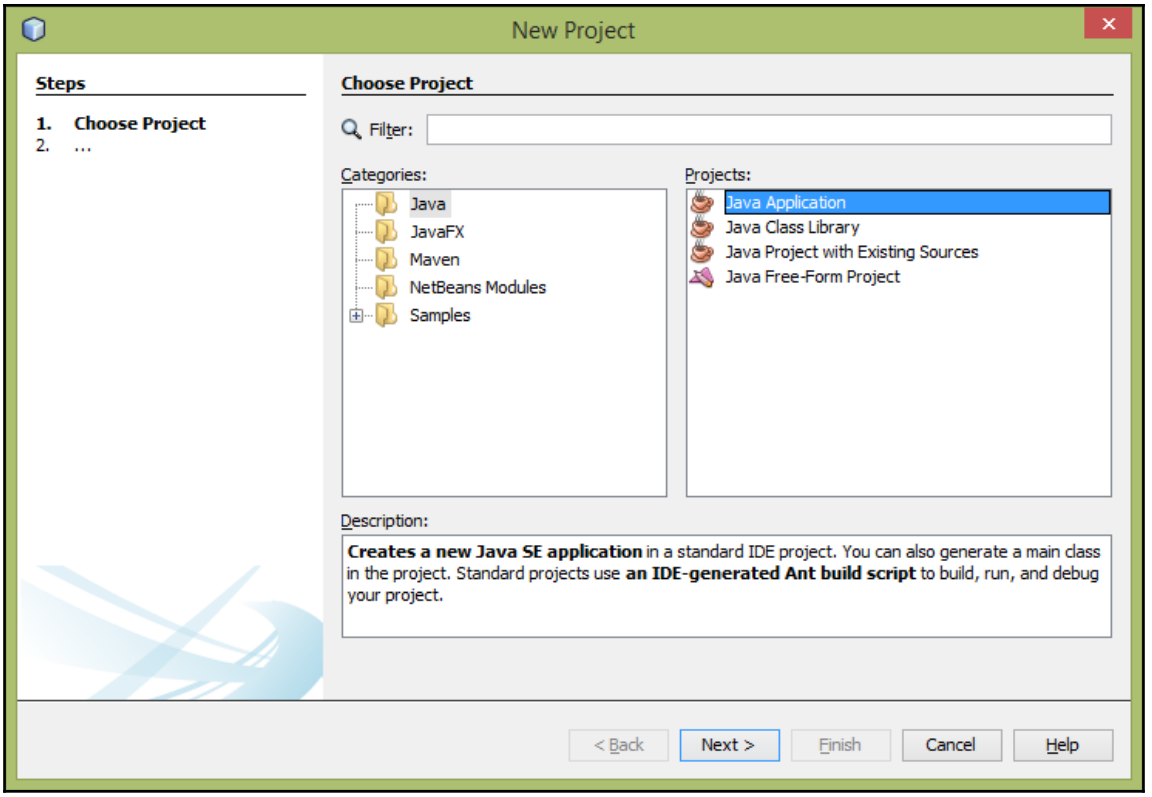


Figure 5-20: Creating ZumoRobot project with NetBeans IDE - step 1. Image credits: Courtesy of Adafruit (`https://www.adafruit.com/`)

2. In the **Name and Location** step, type `ZumoRobot` for the **Project Name** and `com.packt.B05688.chapter5.ZumoRobot` for the **Create Main Class**. Click the **Finish** button to create the project:

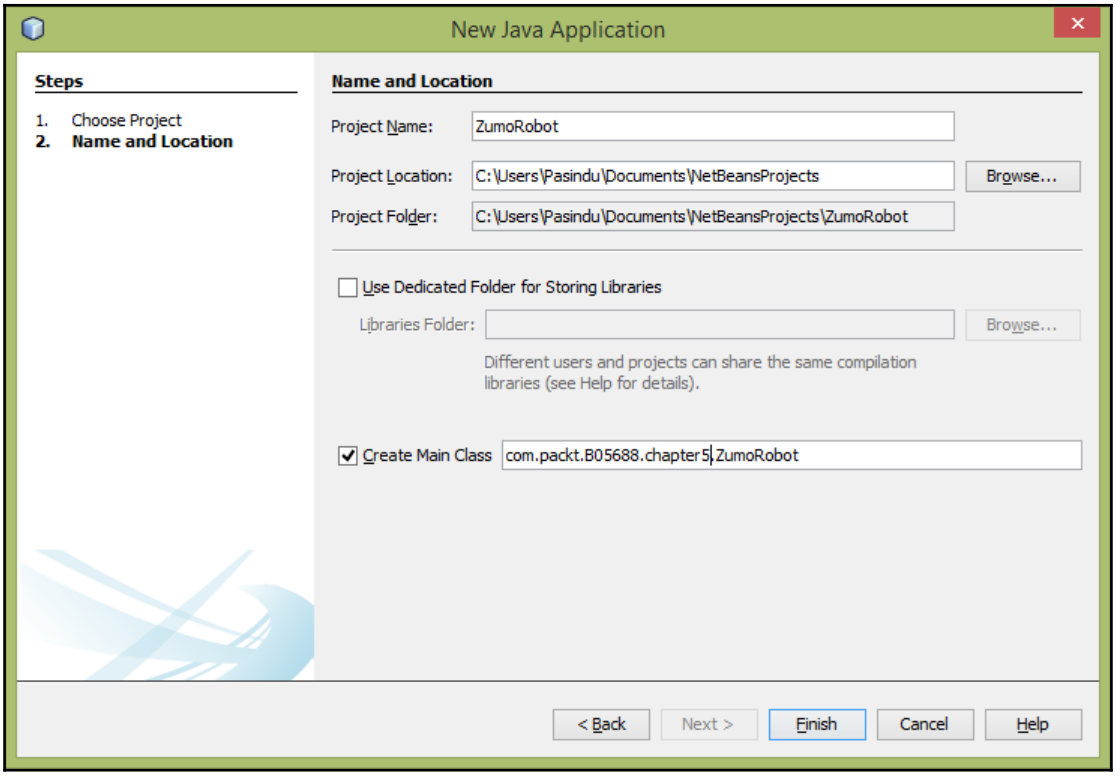


Figure 5-21: Creating ZumoRobot project with NetBeans IDE - step 2. Image credits: Courtesy of Adafruit (`https://www.adafruit.com/`)

3. The NetBeans IDE will create a file named `ZumoRobot.java` and the project environment for you.
4. To work with Raspberry Pi GPIO pins, you should add the `Pi4j` library to your project as discussed in `Chapter 1`, *Setting up Your Raspberry Pi*.
5. Listing 5-1 presents the complete Java program that can be used to control the Zumo robot with the command line. Type or copy all the lines into the `ZumoRobot.java` file.

Listing 5-1: `ZumoRobot.java`

```
package com.packt.B05688.chapter5;
```

```
/*Java Core*/
import java.util.*;
import java.io.*;
/*End Java Core*/

/*Pi4J*/
import com.pi4j.wiringpi.Gpio;
import com.pi4j.wiringpi.SoftPwm;
/*End Pi4J*/

/**
*
* @author Pradeeka
*/
public class ZumoRobot {

/*Variables for motor control pins*/
private static final int LEFT_FORWARD = 1; //GPIO
pin 18
equivalent to WiringPi pin 1
private static final int LEFT_BACKWARD = 4;  //GPIO pin 23
  equivalent to WiringPi pin 4
  private static final int RIGHT_FORWARD = 5; //GPIO pin 24
  equivalent to WiringPi pin 5
  private static final int RIGHT_BACKWARD = 6; //GPIO pin 25
  equivalent to WiringPi pin 6
  /*End Variables for motor control pins*/

  public static void main(String[] args) throws IOException {

        Gpio.wiringPiSetup();

        /*Initializing GPIO pins*/
         //left motor forward
         SoftPwm.softPwmCreate(LEFT_FORWARD, 0, 100); //GPIO 18
        equivalent to WiringPi 1
         //left motor reverse
         SoftPwm.softPwmCreate(LEFT_BACKWARD, 0, 100); //GPIO
23
        equivalent to WiringPi 4
         //right motor forward
        SoftPwm.softPwmCreate(RIGHT_FORWARD, 0, 100); //GPIO 24
        equivalent to WiringPi 5
         //left motor reverse
         SoftPwm.softPwmCreate(RIGHT_BACKWARD, 0, 100); //GPIO
25
        equivalent to WiringPi 6
         /*End Initializing GPIO pins*/
```

```
        Scanner keyboard = new Scanner(System.in);

        boolean quit = false;

        //Scanning input until a 'q' is pressed
         while (!q) {
         System.out.print("Play Zumo (Press h for help): ");
         char move = keyboard.next().charAt(0);

        switch (move) {
         case 'w':
         System.out.println("forwading...");
         forward();// move forward
         break;
         case 'z':
         System.out.println("reversing...");
         backward();// move backward
         break;
         case 'a':
         System.out.println("point turning to left...");
         pointTurnLeft(); // point turn left
         break;
         case 's':
         System.out.println("point turning to right...");
         pointTurnRight(); // point turn right
         break;
         case 'x':
         System.out.println("forward swing turning to
left...");
         forwardSwingTurnLeft(); // forward swing turn left
         break;
         case 'c':
         System.out.println("backward swing turning to
left...");
         backwardSwingTurnLeft(); // backward swing turn left
         break;
         case 'n':
         System.out.println("forward swing turning to
right...");
         forwardSwingTurnRight(); // forward swing turn right
         break;
         case 'm':
         System.out.println("backward turning to right...");
         backwardSwingTurnRight(); // backward swing turn right
         break;
         case 'h':
         System.out.println("Starting help...");
         help(); // help
```

```
 break;
 case 'q':
 quit = true; // quit playing
 break;
 default:
 System.out.println("That is not a valid command!");
 }
 }
 }

 /*Forward*/
 public static void forward() {
 SoftPwm.softPwmWrite(LEFT_FORWARD, 100);
 SoftPwm.softPwmWrite(RIGHT_FORWARD, 100);
 }
/*End Forward*/

 /*Backward*/
 public static void backward() {
 SoftPwm.softPwmWrite(LEFT_BACKWARD, 100);
 SoftPwm.softPwmWrite(RIGHT_BACKWARD, 100);
 }
 /*End Reverse*/

 /*Point Turn*/
/*Turn Left*/
 public static void pointTurnLeft() {
 SoftPwm.softPwmWrite(LEFT_BACKWARD, 100);
 SoftPwm.softPwmWrite(RIGHT_FORWARD, 100);
 }
/*End Turn Left*/

 /*Turn Right*/
  public static void pointTurnRight() {
  SoftPwm.softPwmWrite(LEFT_FORWARD, 100);
  SoftPwm.softPwmWrite(RIGHT_BACKWARD, 100);
  }
 /*End Turn Right*/

/*Swing Turn*/
/*Forward Left Swing Turn*/
 public static void forwardSwingTurnLeft() {
 SoftPwm.softPwmWrite(LEFT_FORWARD, 0);
 SoftPwm.softPwmWrite(RIGHT_FORWARD, 100);
 }
/*End Forward Left Swing Turn*/

/*Backward Left Swing Turn*/
```

```
public static void backwardSwingTurnLeft() {
SoftPwm.softPwmWrite(LEFT_FORWARD, 0);
SoftPwm.softPwmWrite(RIGHT_BACKWARD, 100);
}
/*End Backward Left Swing Turn*/

/*Forward Right Swing Turn*/
 public static void forwardSwingTurnRight() {
 SoftPwm.softPwmWrite(LEFT_FORWARD, 100);
 SoftPwm.softPwmWrite(RIGHT_FORWARD, 0);
 }
/*End Forward Right Swing Turn*/

/*Backward Right Swing Turn*/
public static void backwardSwingTurnRight() {
SoftPwm.softPwmWrite(LEFT_BACKWARD, 100);
SoftPwm.softPwmWrite(RIGHT_FORWARD, 0);
}
/*End Backward Right Swing Turn*/

/*Stop*/
 public static void stop() {
 SoftPwm.softPwmWrite(LEFT_FORWARD, 0);
 SoftPwm.softPwmWrite(LEFT_BACKWARD, 0);
 SoftPwm.softPwmWrite(RIGHT_FORWARD, 0);
 SoftPwm.softPwmWrite(RIGHT_BACKWARD, 0);
 }
/*End Stop*/

/*Help*/
public static void help() {
System.out.println("w -> Move Forward.");
System.out.println("z -> Move Backward.");
System.out.println("a -> Point Turn Left.");
System.out.println("s -> Point Turn Right.");
System.out.println("x -> Forward Swing Turn Left.");
System.out.println("c -> Backward Swing Turn Left.");
System.out.println("n -> Forward Swing Turn Right.");
System.out.println("m -> Backward Swing Turn Right.");
System.out.println("h -> Help.");
}
 /*End Help*/
}
```

Let's examine the most important sections in the `ZumoRobot.java` file.

6. First import all the necessary Java packages and classes into your program. The `Gpio` and `SoftPwm` classes of the `WiringPi` library allow you to enable PWM on GPIO pins. This is very useful because Raspberry Pi has only a single GPIO pin with PWM enabled by default, which is GPIO pin 18:

```
/*Java Core*/
import java.util.*;
import java.io.*;
/*End Java Core*/

/*Pi4J*/
import com.pi4j.wiringpi.Gpio;
import com.pi4j.wiringpi.SoftPwm;
/*End Pi4J*/
```

7. The following variables hold the `WiringPi` pin numbers that are equivalent to GPIO pins:

```
/*Variables for motor control pins*/
private static final int LEFT_FORWARD = 1; //GPIO pin 18
equivalent to WiringPi pin 1
private static final int LEFT_BACKWARD = 4; //GPIO pin 23
equivalent to WiringPi pin 4
private static final int RIGHT_FORWARD = 5; //GPIO pin 24
equivalent to WiringPi pin 5
private static final int RIGHT_BACKWARD = 6; //GPIO pin 25
equivalent to WiringPi pin 6
/*End Variables for motor control pins*/
```

Table 5-3 presents all the GPIO pins that are equivalent to the WiringPi pins:

WiringPi	**GPIO**
8	2
9	3
7	4
0	17
2	27
3	22
12	10

13	9
14	11
21	5
22	6
23	13
24	19
25	26
15	14
16	15
1	18
4	23
5	24
6	25
10	8
11	7
26	12
27	16
28	20
29	21

Table 5-3: WiringPi and GPIO pin mapping

8. The `wiringPiSetup` method initializes the `WiringPi` library, which is needed for PWM:

```
Gpio.wiringPiSetup();
```

9. The `softPwmCreate` method creates a software-controlled PWM pin. The method signature should be as follows:

```
int softPwmCreate (int pin, int initialValue, int pwmRange) ;
```

- `pin`**:** The GPIO pin to use as a PWM pin.
- `value`**:** The value to initialize the PWM pin (between `0` (off) to `100` (fully on)).
- `range`**:** The maximum range. Use `100` for the `pwmRange`.

You can use any GPIO pin and the pin numbering should be equivalent to the `wiringPiSetup` (see WiringPi pin assignments for more information). Use 100 for the `pwmRange`, then the value at `initialValue` can be anything from `0` (off) to `100` (fully on) for the given pin.

The method returns `0` for success and anything else indicates an error.

The following code snippet creates software-controlled PWM pins on GPIO pins 18, 23, 24, and 25. The equivalent WiringPi pins are 1, 4, 5, and 6 respectively:

```
/*Initializing GPIO pins*/
 //left motor forward
 SoftPwm.softPwmCreate(LEFT_FORWARD, 0, 100); //GPIO 18 equivalent
to WiringPi 1
 //left motor reverse
 SoftPwm.softPwmCreate(LEFT_BACKWARD, 0, 100); //GPIO 23 equivalent
to WiringPi 4
 //right motor forward
 SoftPwm.softPwmCreate(RIGHT_FORWARD, 0, 100); //GPIO 24 equivalent
to WiringPi 5
 //left motor reverse
 SoftPwm.softPwmCreate(RIGHT_BACKWARD, 0, 100); //GPIO 25
equivalent to WiringPi 6
 /*End Initializing GPIO pins*/
```

10. The `softPwmWrite` method can be used to update the PWM value on the given pin. The method signature should be as follows:

```
void softPwmWrite (int pin, int value) ;
```

- `pin`**:** The GPIO pin to use as a PWM pin
- `value`**:** The value to initialize the PWM pin (between `0` (off) to `100` (fully on))

The following code snippet shows the use of the `softPwmWrite` method to move the Zumo robot forward. The value `100` is used to drive the motor at full speed. To turn off the motor, you can use the value `0`:

```
public static void forward() {
SoftPwm.softPwmWrite(LEFT_FORWARD, 100);
SoftPwm.softPwmWrite(RIGHT_FORWARD, 100);
}
```

11. `ZumpRobot.java` includes a set of user-defined methods such as `forward`, `backward`, `pointTurnLeft`, `pointTurnRight`, `forwardSwingTurnLeft`, `backwardSwingTurnLeft`, `forwardSwingTurnRight`, and `backwardSwingTurnRight` to move the robot. A `switch` statement can be used to select the correct method by the pre-configured key from the keyboard input when movement is needed.

Table 5-4 shows the keyboard mapping with all the methods that can be used to play with the Zumo robot by pressing the keyboard keys.

Keyboard key	**Method**	**Description**
w	`forward`	Move forward
z	`backward`	Move backward
a	`pointTurnLeft`	Point turn left
s	`pointTurnRight`	Point turn right
x	`forwardSwingTurnLeft`	Forward swing turn left
c	`backwardSwingTurnLeft`	Backward swing turn left
n	`forwardSwingTurnRight`	Forward swing turn right
m	`backwardSwingTurnRight`	Backward swing turn right
h	`help`	Help

Table 5-4: Keyboard mapping to methods

Running and testing your Java program

First, insert four rechargeable NiMH AA batteries into the battery compartment of the Zumo chassis. Then take the USB battery pack, shown in Figure 5-22, to power your Raspberry Pi:

Figure 5-22: USB battery pack for Raspberry Pi - 10000mAh - 2 x 5V outputs

The ROMOSS USB battery pack has two USB A ports for regulated 5V output. The port placed on top can provide 1 A and the bottom one can provide 2 A. Make sure to use the 2 A USB port marked with *double flash icons* to power the Raspberry Pi (Figure 5-23). Now connect the USB battery pack to the Raspberry Pi using the micro USB cable:

Figure 5-23: 2A output marked with double flash icons

1. Using the NetBeans IDE, build the Java project `ZumoRobot` by clicking **Run|Bulid Project (ZumoRobot)** in the menu bar or pressing F11 on your keyboard. Fix any errors you encountered during the build process. After successfully building the project, it's time to test your `ZumoRobot` wirelessly with the PuTTY terminal.
2. Open your PuTTY terminal by providing the IP address of your Raspberry Pi and the port, which is 22. Then run `ZumoRobot.jar`, located in the SD card of the Raspberry Pi from the command line with `sudo`:

```
sudo java –jar
/home/pi/RASPI3JAVA/ZumoRobot/dist/ZumoRobot.jar
```

You will get an output and prompt for the keyboard input similar to that shown in Figure 5-24:

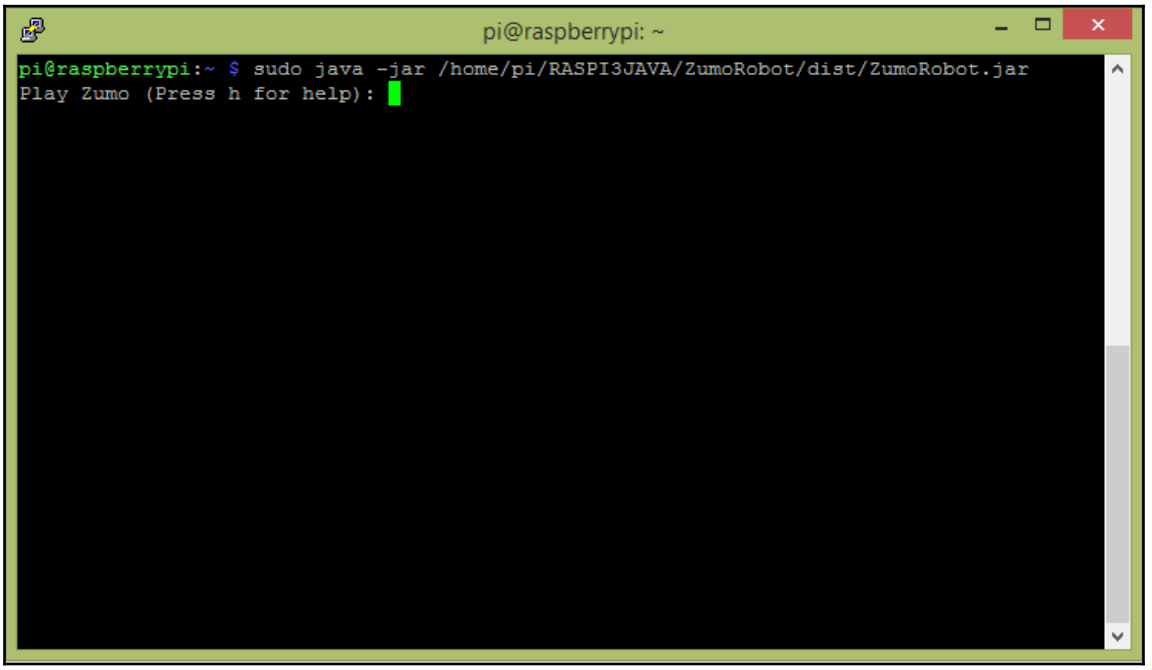


Figure 5-24: Running ZumoRobor.jar using PuTTY

3. Press the correct keys on your keyboard to choose the correct operation, as presented in table 5-1. Figure 5-25 shows some of the operations that can be performed through the PuTTY terminal:

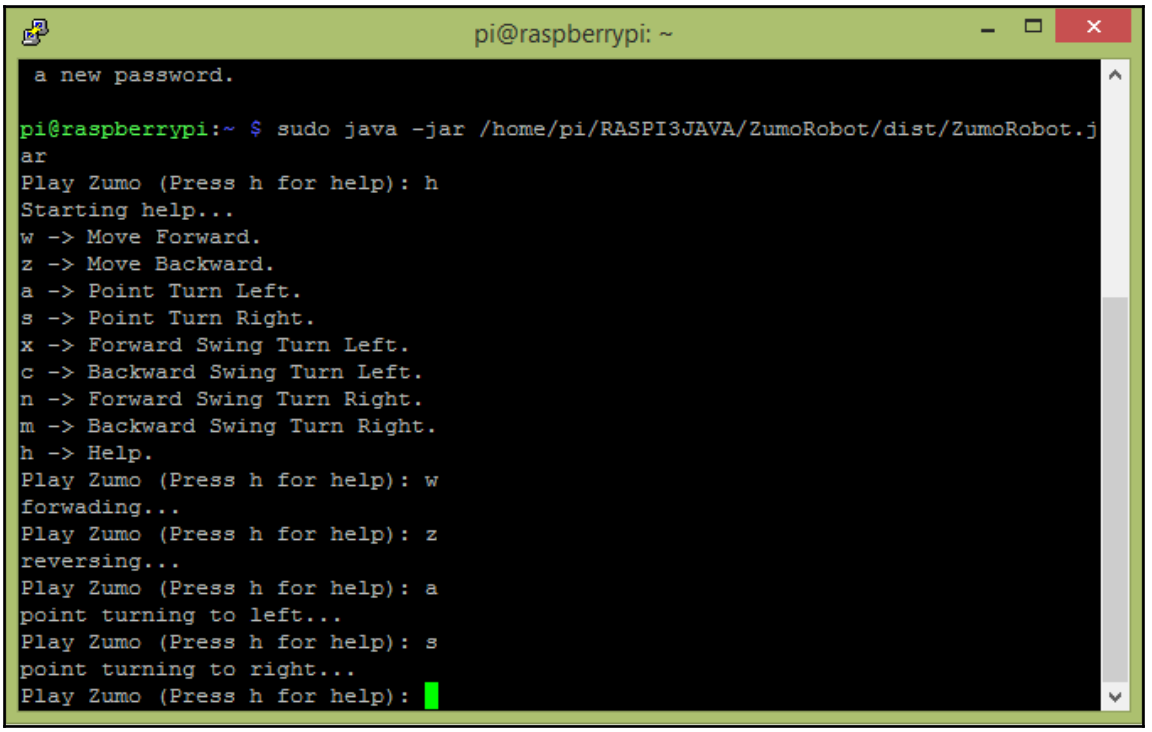


Figure 5-25: Operating Zumo Robot with keyboard inputs

The operating range of the Zumo wireless robot is dependent on the signal strength of your Wi-Fi network. However, you can connect an external Wi-Fi dongle with a high-gain antenna to the Raspberry Pi to receive weak signals and perform long-range operations:

Summary

In this chapter, you have learned how to build a wirelessly-controlled robot with the Zumo chassis kit. The quick learning curve presented how to assemble the Zumo chassis kit with the Raspberry Pi 3 and a motor driver to drive two gear motors and form the engine. Also, you connected to the robot using the PuTTY terminal installed on your computer with SSH. The built-in Wi-Fi module of the Raspberry Pi 3 allowed you to connect wirelessly from your computer, and then you execute the `jar` file located in the SD card of the Pi through SSH, using the keyboard.

You can connect an external Wi-Fi antenna to the Raspberry Pi 3 to achieve long-range connectivity with your Wi-Fi network, but it will violate FCC regulations.

The `Chapter 6`, *Building a Multipurpose IoT Controller*, shows how to build a multipurpose IoT controller with simple web services and a SQLite database. This project allows you to gather and store sensor data in your local database, visualize it in your own web interface, and control the Raspberry Pi from the web interface.

6
Building a Multipurpose IoT Controller

Raspberry Pi is great for building powerful server-based applications, configuring it to act as a web server by installing a web server such as Nginx, Apache, Jetty, or lighttpd. A web server can serve a web application to many users concurrently through a network. But unlike other traditional web applications, Raspberry Pi-hosted web applications can be used to interact with the Raspberry Pi itself, and other peripherals attached to it.

The following diagram shows a Raspberry Pi board configured as a web server with two clients connected to it through a router:

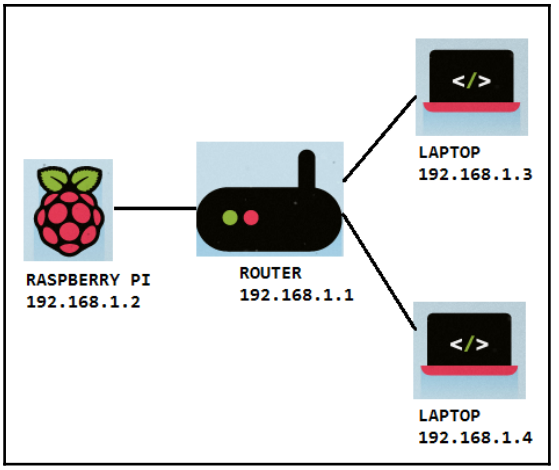


Figure 6-1: Simple Raspberry Pi client-server model

A lightweight web application can be used to interact with sensors and actuators attached to the Raspberry Pi. As an example, you could control an LED attached to a GPIO pin or display readings of a sensor attached to the I2C pins of the Raspberry Pi.

In this chapter, you will:

- Install and configure Jetty servlet engine on the Raspberry Pi to convert it into a web server
- Install and configure the Jetty server on NetBeans IDE with the Jetty server plugin for Maven in NetBeans
- Develop Maven Project with Archetype using NetBeans
- Deploy a **Web Application ARchive** (**WAR**) file to the Raspberry Pi
- Control an LED attached to the GPIO pin through the web user interface

Prerequisites

To build the series of projects that will be present in this chapter, you will need following things in your tool box:

- Raspberry Pi 3 board
- 3 mm LED
- 220 Ohm resistor
- Hookup wires

Preparing your Raspberry Pi board

Before going in too deep, let's prepare our Raspberry Pi board by installing the Jetty server for projects that will be discussed in this chapter.

Installing and configuring Jetty servelet engine

Jetty servlet engine is a lightweight server component that can be used to host web applications. Jetty servlet engine consumes less memory than an Apache server and it is great for saving resources on the Raspberry Pi.

You can learn more about Jetty by visiting `http://www.eclipse.org/jetty/`. It includes documentation, downloads, a Maven plugin, Eclipse tooling, mailing lists, Javadocs, tools, blogs, and much more that can help you to learn to use Jetty in your projects.

The following steps will guide you through installing and configuring it on your Raspberry Pi, followed by instructions for a simple web application that can be developed and run to verify the Jetty service:

1. The official Jetty page (Figure 6-2) at `http://www.eclipse.org/jetty/download.html`includes download links for all the latest minor releases. The latest release of Jetty for **JDK 8+** is `9.4.2.v20170220` at the time of writing this book, and you need the `.zip` file, which is located at `http://central.maven.org/maven2/org/eclipse/jetty/jetty-distribution/9.4.2.v20170220/jetty-distribution-9.4.2.v20170220.zip`. You can download the `.zip` file directly onto the Raspberry Pi with the `wget` command. The download link can be easily obtained by right-clicking on the download label and selecting **Copy** link address from the context menu:

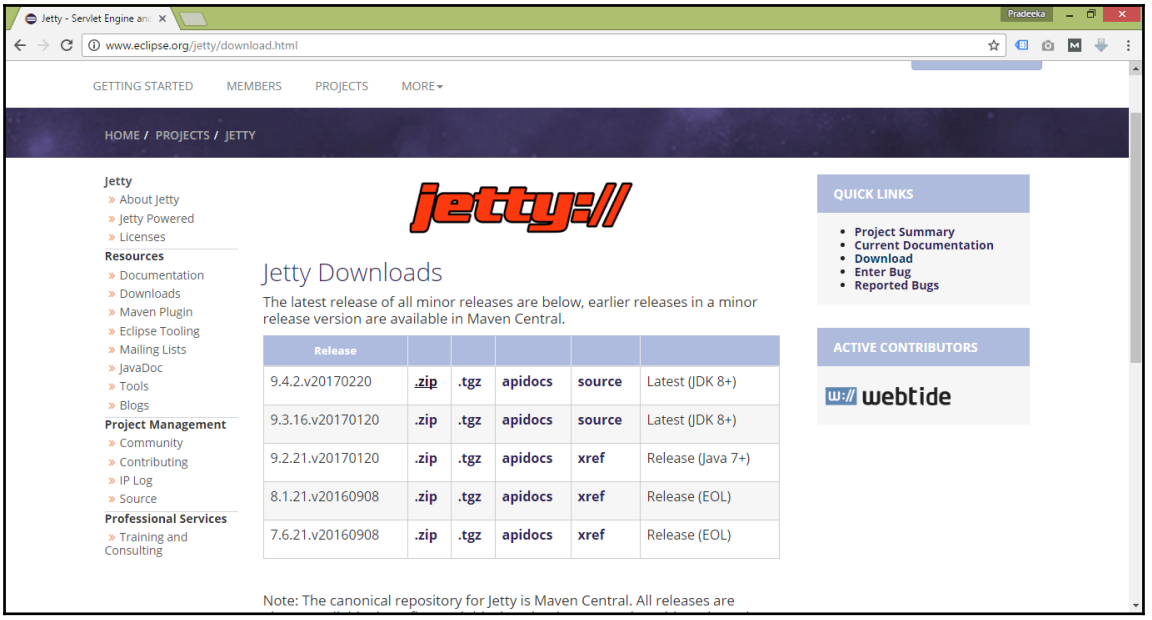


Figure 6-2: Jetty download page

2. As a best practice, update and upgrade your Raspberry Pi using the following commands:

```
sudo apt-get update
sudo apt-get upgrade
```

3. Download the `jetty-distribution-9.4.2.v20170220.zip` file onto the Raspberry Pi with the `wget` command:

```
wget http://central.maven.org/maven2/org/eclipse/
jetty/jetty-distribution/9.4.2.v20170220/jetty-
distribution-9.4.2.v20170220.zip
```

4. After that, you have to extract the downloaded `.zip` file with the `unzip` command. A directory will be created in the same location, in this case `jetty-distribution-9.4.2.v20170220`:

```
unzip jetty-distribution-9.4.2.v20170220.zip
```

5. The `jetty-distribution-9.4.2.v20170220` directory contains a file named `start.jar`. The `start.jar` file helps you to start the Jetty server as a standalone server on the Raspberry Pi. This is the most basic way to start the Jetty server. Now simply navigate to the `jetty-distribution-9.4.2.v20170220` directory and run the `start.jar` file:

```
java -jar start.jar
```

6. The Jetty server will start and you will get output similar to the following in the console. The default port number of the Jetty server is `8080`:

```
2017-03-09 21:49:20.891:INFO::main: Logging
initialized @3687ms to
org.eclipse.jetty.util.log.StdErrLog

2017-03-09
21:49:22.420:WARN:oejs.HomeBaseWarning:main: This
instance of Jetty is not running from a separate
{jetty.base} directory, this is not recommended.
Seedocumentation at
http://www.eclipse.org/jetty/documentation/current/startu
        p.html

2017-03-09 21:49:22.723:INFO:oejs.Server:main: jetty-
9.4.2.v20170220
```

```
2017-03-09
21:49:22.908:INFO:oejdp.ScanningAppProvider:main:
Deployment monitor [file:///home/pi/jetty-
distribution-9.4.2.v20170220/webapps/] at interval 1
2017-03-09
21:49:23.065:INFO:oejs.AbstractConnector:main: Started
ServerConnector@10a2abb{HTTP/1.1,[http/1.1]}
{0.0.0.0:8080}
2017-03-09 21:49:23.071:INFO:oejs.Server:main: Started
@5868ms
```

7. Now access the Jetty server with the IP address of the Raspberry Pi and the port number associated with the Jetty service. As an example, the Jetty home page can be access by typing the following URL into your browser:

```
http://192.168.1.2:8080
```

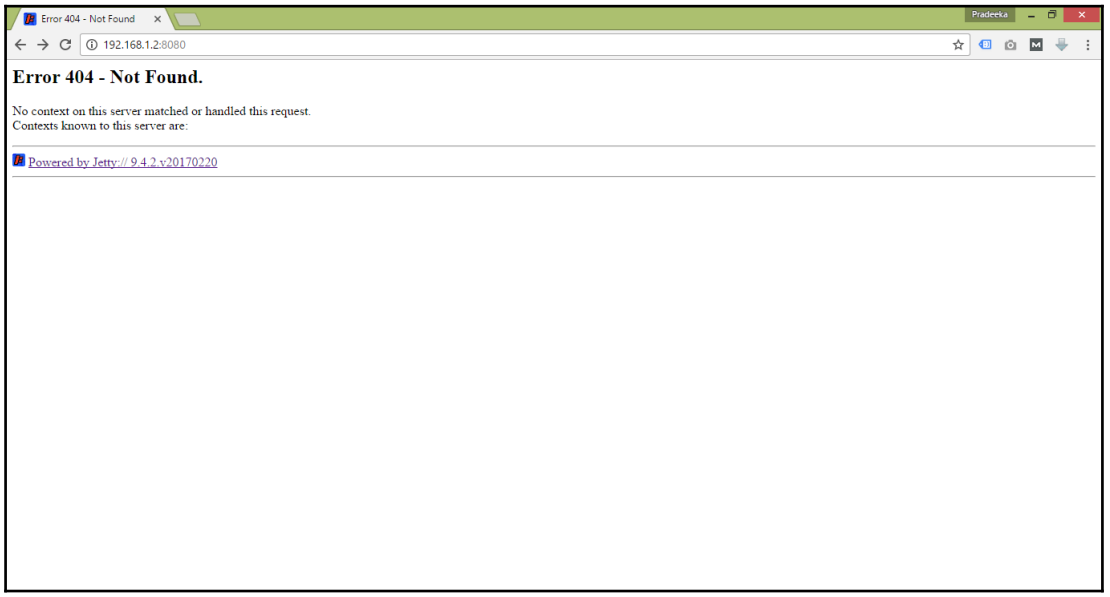


Figure 6-3: Jetty server start page

You can stop the Jetty server, stopping the service, by simply pressing *Ctrl + C* from the terminal window.

The Jetty server responds with `Error 404 - Not Found`, because you still haven't deployed any web applications to it.

In the next section, you will learn how to write a Java web application using servlets and deploy it to the Jetty server that is running on the Raspberry Pi.

Writing your first Java web application

The servlet you will be writing to control the Raspberry Pi's GPIO pin through the internet or any local area network requires a series of configurations to build and test in the Windows environment. Then you will deploy the project to the Raspberry Pi manually and run it on the Jetty server.

Creating a Maven project from Archetype

1. Open the NetBeans IDE and from the menu bar, click on **File** | **New Project**. The **New Project** wizard will appear.
2. In **Choose Project** (step 1), select **Maven** under **Categories** and **Project from Archetype** under **Projects**,as shown in Figure 6-4.
3. Then, click the **Next** button to proceed:

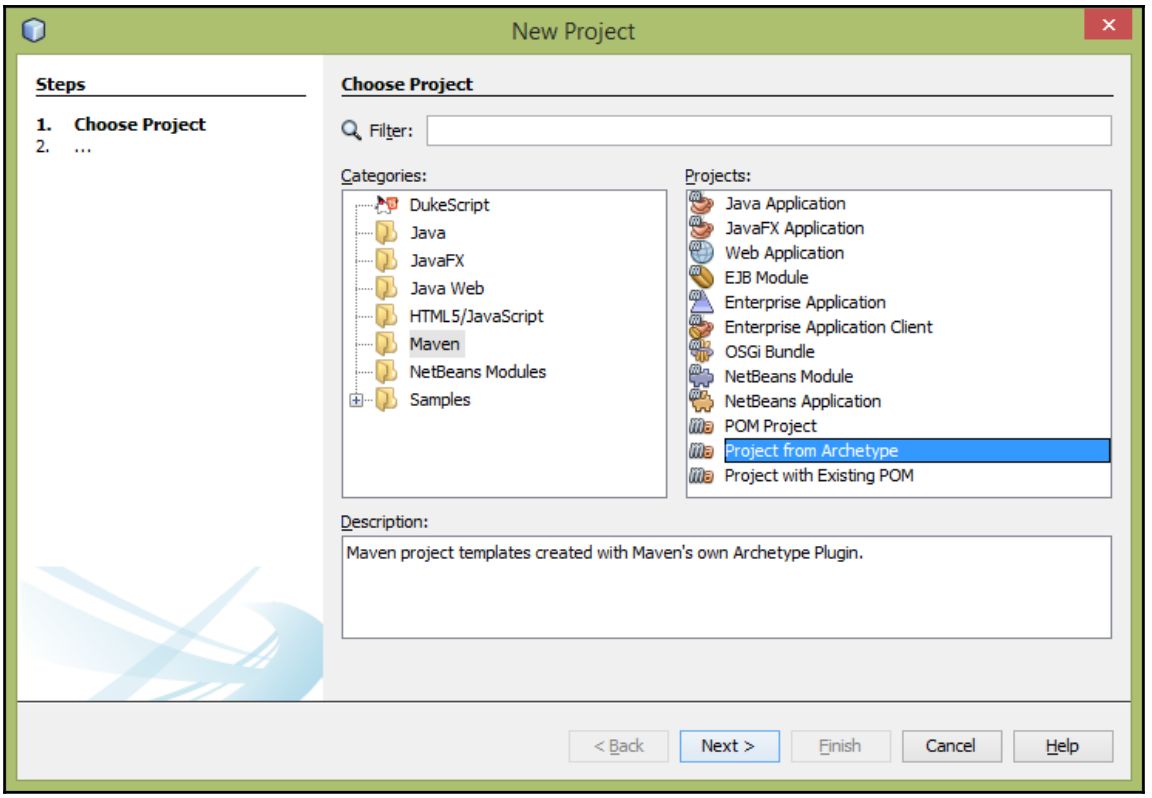


Figure 6-4: Choose project

4. In **Maven Archetype** (step 2), select **maven-archetype-webapp** under **Known Archetypes**, as shown in Figure 6-5.

5. Then, click the **Next** button to proceed:

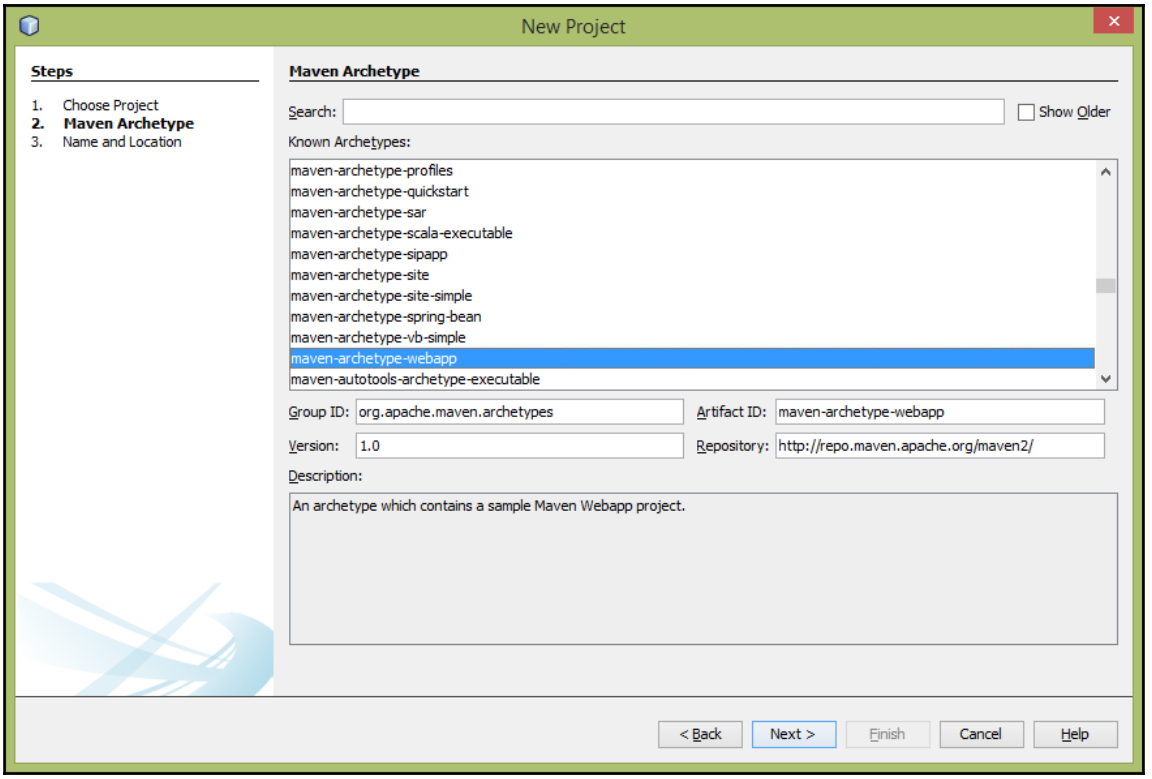


Figure 6-5: Maven Archetype

6. In **Name and Location** (step 3), type `iot` for **Project Name**, `com.packt.B05688` for **Group Id**, and `com.packt.B05688.chapter6` for **Package**, as shown in Figure 6-6.

7. Then, click the**Finish** button to create the project:

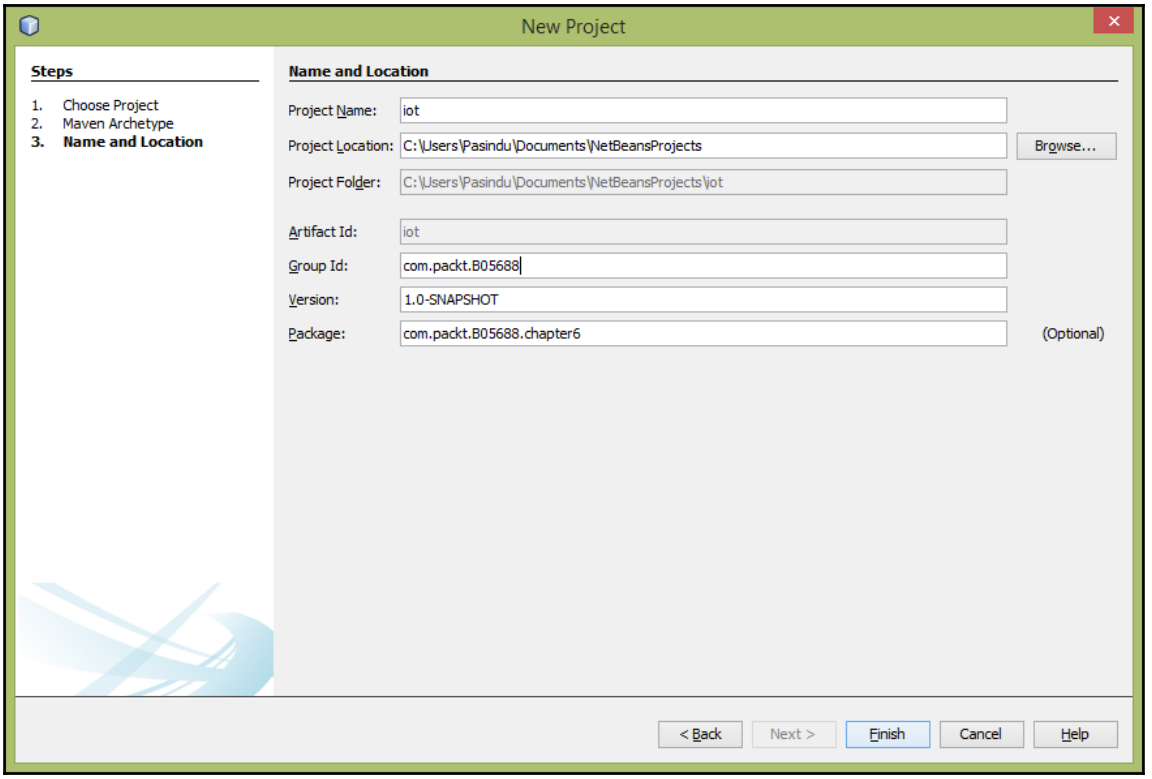


Figure 6-6: Name and Location

8. While creating the project by NetBeans, you will get output in the **Output** window similar to that shown in Figure 6-7:

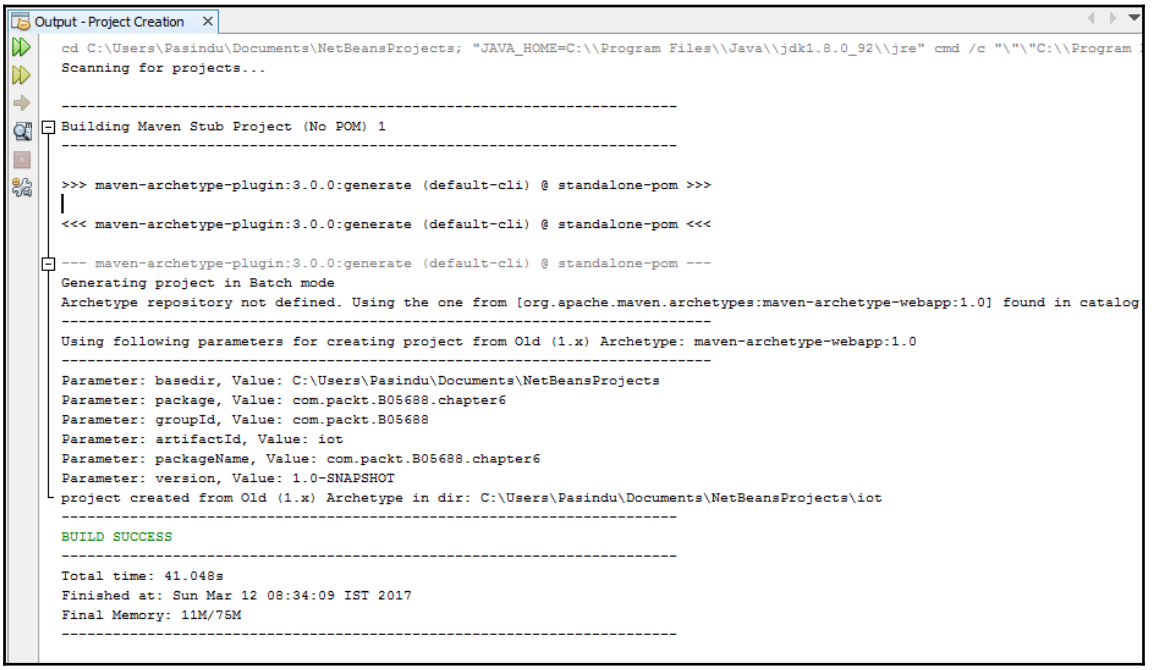


Figure 6-7: Output window

9. You can find the newly created `iot Maven Webapp` project in the **Projects explorer** (Figure 6-8). Both `web.xml` and `pom.xml` are very important files that can be used to further configure your project:

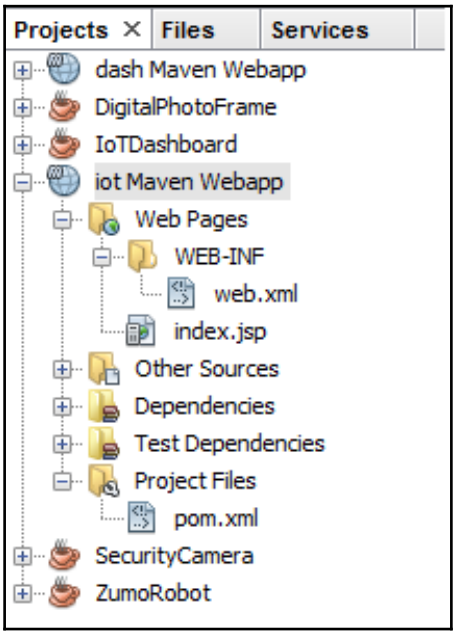


Figure 6-8: Project explorer

10. Now open `pom.xml` from the **Projects explorer** by double-clicking on it. The contents of the `pom.xml` file are shown in Listing 6-1:

Listing 6-1: `pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.packt.B05688</groupId>
 <artifactId>iot</artifactId>
 <packaging>war</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>iot Maven Webapp</name>
 <url>http://maven.apache.org</url>
 <dependencies>
 <dependency>
```

```
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>
 </dependencies>
 <build>
 <finalName>iot</finalName>
 </build>
</project>
```

11. Now you're going to add the Jetty plugin to `pom.xml`, which helps you to start the Jetty server automatically and deploy your project with NetBeans. To configure the `Jetty` plugin, add the code shown in Listing 6-2 to `pom.xml`. The new code block is highlighted in bold; make sure to add it inside the `<build></build>` tags:

    Listing 6-2: `pom.xml`

```
  <project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.packt.B05688</groupId>
 <artifactId>iot</artifactId>
 <packaging>war</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>iot Maven Webapp</name>
 <url>http://maven.apache.org</url>
     <dependencies>
      <dependency>
      <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>3.8.1</version>
      <scope>test</scope>
     </dependency>
     </dependencies>
     <build>
     <finalName>iot</finalName>

<plugins>
     <plugin>

 <groupId>org.mortbay.jetty</groupId>
```

```
<artifactId>maven-jetty-plugin</artifactId>

<version>6.1.14</version>

<configuration>

<stopPort>9966</stopPort>

<stopKey>jetty-stop</stopKey>

<scanIntervalSeconds>5</scanIntervalSeconds>

</configuration>

</plugin>

</plugins>

</build>
</project>
```

12. Then, right-click on the **iot Maven Webapp** project in the**Projects explorer**, and from the context menu click **Properties**. In the **Project Properties** dialog box, under **Categories**, click **Actions**. Now do the following:
    - In the *Actions* list box, click on **Run Project**.
    - In the **Execute Goals** text box, type `jetty:stop jetty:run`

13. Click the **OK** button to save the settings:

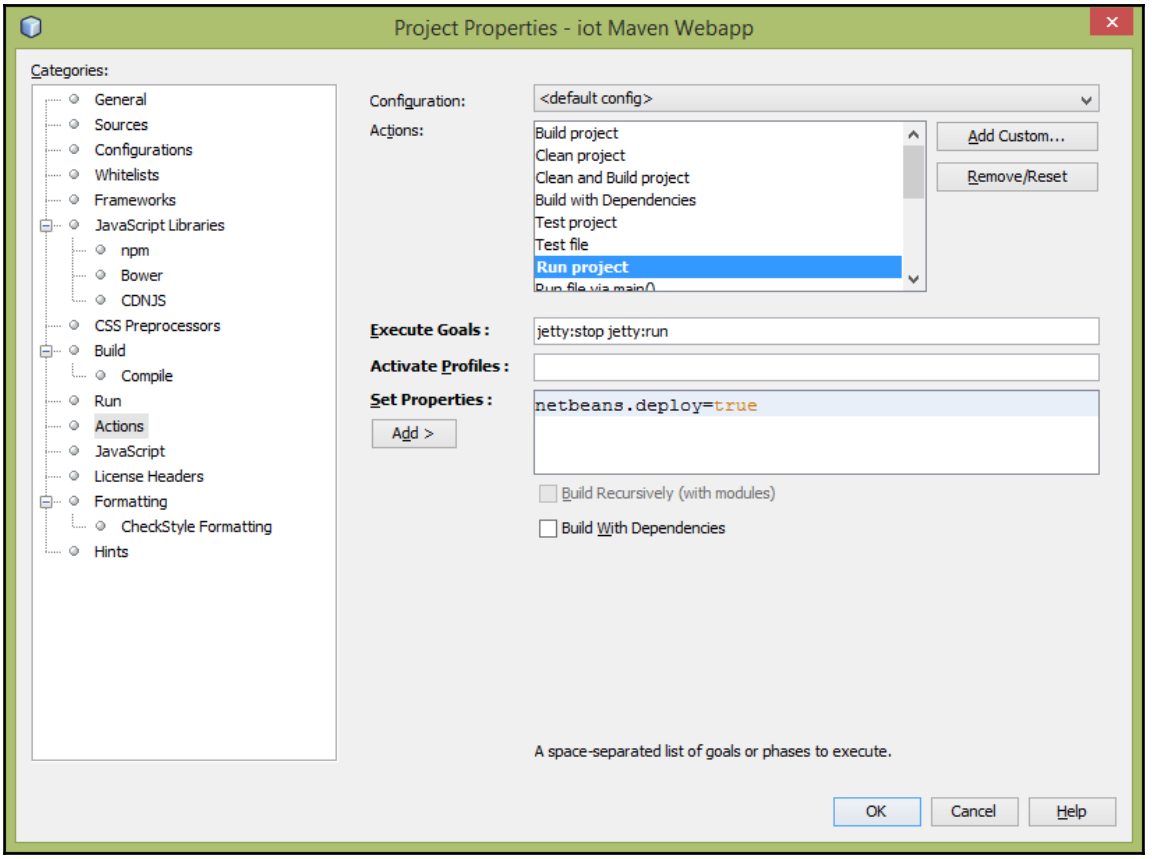


Figure 6-9: Project properties

14. It's time to build and run your project. Right-click on the **iot Maven Webapp** project in the **Projects explorer** and from the context menu click on **Build**. After successfully building the project, click on **Run** from the context menu to run the project. This will start the Jetty server on the default port number 8080 and deploy the web application to the server. Your web application is available at `http://localhost:8080/iot` and you can access it using your web browser. You will get the output (Figure 6-10) from index.jps, which is located in the Web Pages directory:

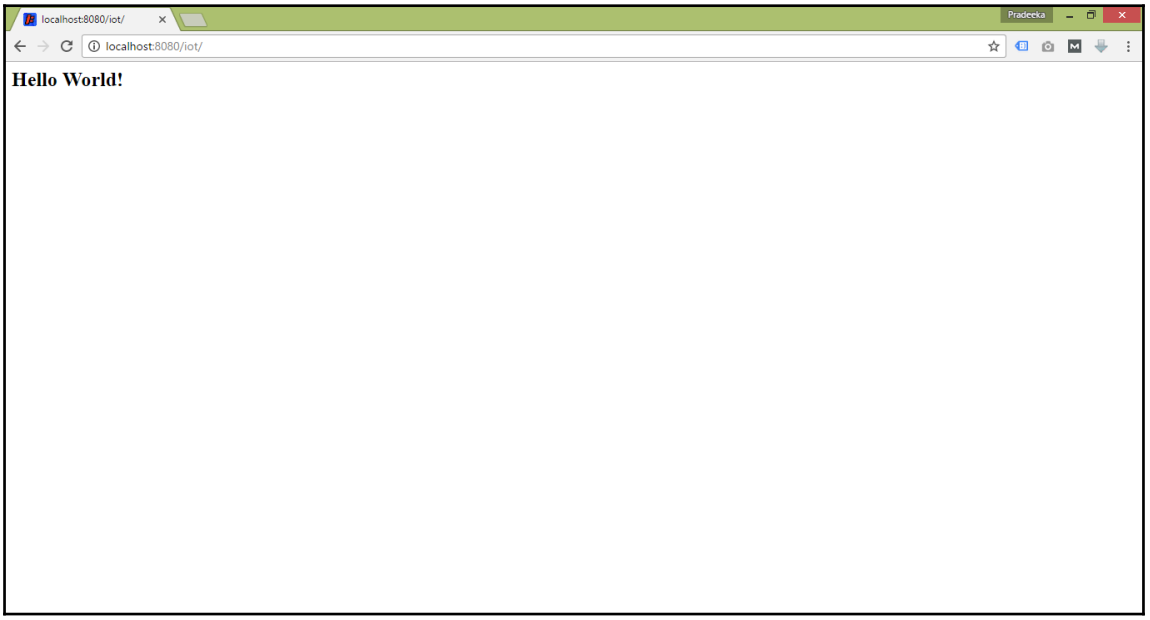


Figure 6-10: Response for web application at http://localhost:8080/iot

Creating a servlet

The Raspberry Pi can be controlled through a simple web page that can be generated by a simple servlet. The following steps will guide you through creating and configuring a servlet with NetBeans:

1. Right-click on the **iot Maven Webapp** in the **Projects explorer** and from the context menu click **New | Servlet**. In the **New Servlet** dialog box (Figure 6-11), type `gpio` as the **File Name**.
2. Click the **Next** button to proceed:

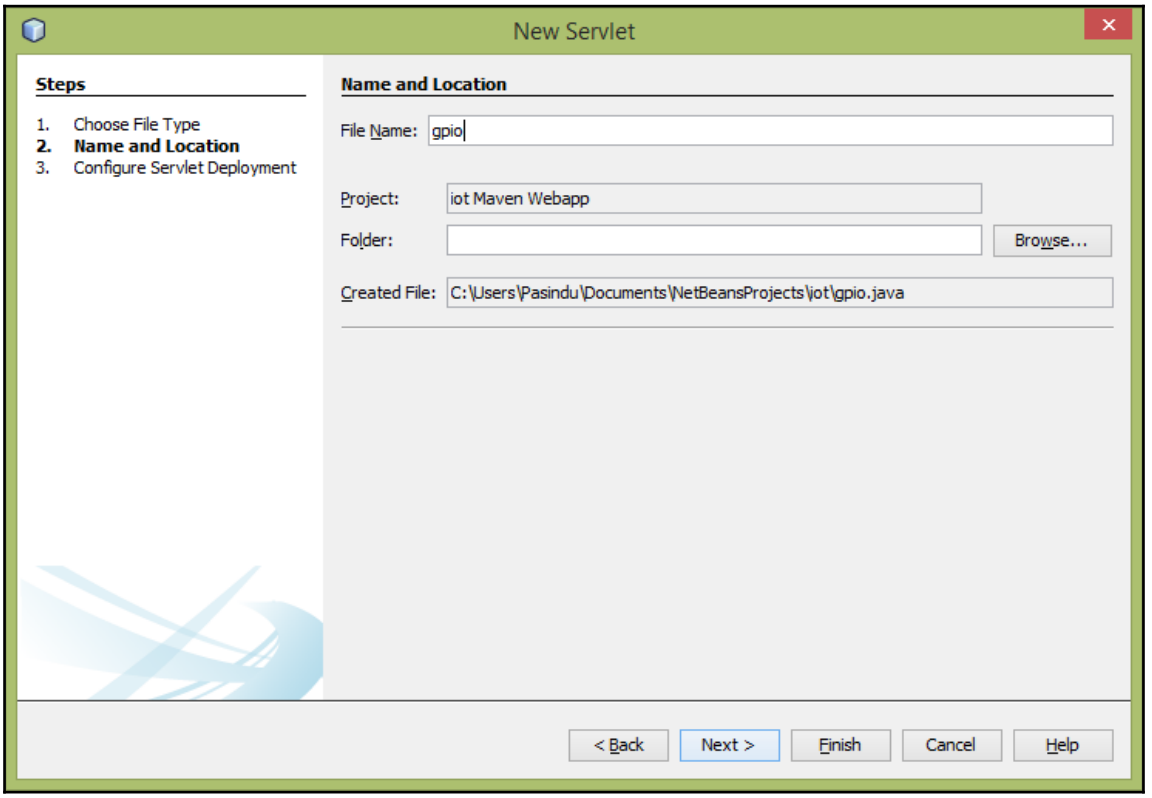


Figure 6-11: New servlet - Name and Location

3. In the final step (Figure 6-12), select **Add information to deployment descriptor (web.xml)**. Don't change the default **Servlet Name** (`gpio`) and **URL Pattern** (`/gpio`). Click the **Finish** button to save the configuration:

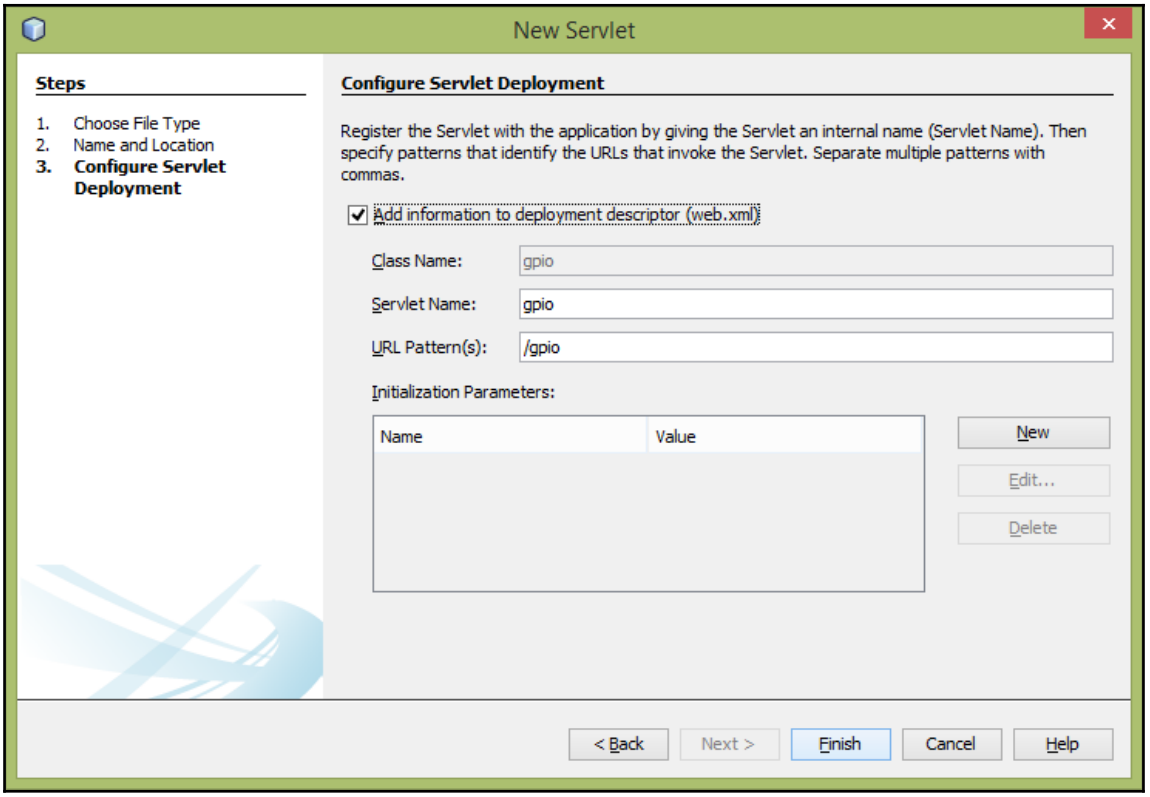


Figure 6-12: New servlet - Configure servlet deployment

4. Now open the `pom.xml` file again and add the code highlighted in bold in Listing 6-3:

Listing 6-3: `pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.packt.B05688</groupId>
 <artifactId>iot</artifactId>
```

```
 <packaging>war</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>iot Maven Webapp</name>
 <url>http://maven.apache.org</url>
 <dependencies>
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>3.8.1</version>
 <scope>test</scope>
 </dependency>

<dependency>

 <groupId>javax.servlet</groupId>

 <artifactId>servlet-api</artifactId>

 <version>2.5</version>

 <scope>provided</scope>

 </dependency>

 <dependency>

 <groupId>javax.servlet.jsp</groupId>

 <artifactId>jsp-api</artifactId>

 <version>2.0</version>

 <scope>provided</scope>

 </dependency>
 </dependencies>
 <build>
 <finalName>iot</finalName>
 <plugins>
 <plugin>
 <groupId>org.mortbay.jetty</groupId>
 <artifactId>maven-jetty-plugin</artifactId>
 <version>6.1.14</version>
 <configuration>
 <stopPort>9966</stopPort>
 <stopKey>jetty-stop</stopKey>
 <scanIntervalSeconds>5</scanIntervalSeconds>
 </configuration>
```

```
 </plugin>
 </plugins>
 </build>
</project>
```

5. Open the `gpio.java` file and replace the existing code with the new code listed in Listing 6-4:

Listing 6-4: `gpio.java`

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;


        public class gpio extends HttpServlet {
        private static final long serialVersionUID = 1L;
        GpioController gpio;
        GpioPinDigitalOutput gpio1;

        public gpio() {

        GpioController gpio = GpioFactory.getInstance();

         gpio1 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01,
"GPIO
                  01",
          PinState.LOW);

         }


         public void doPost(HttpServletRequest request,
         HttpServletResponse response) throws IOException,
        ServletException {
         // get the number of the LED to turn on and print to
console
         String action = request.getParameter("action");
```

```
            System.out.println("action=" +
            request.getParameter("action"));

            try {
            if (action.equals("1")) {
            System.out.println("led is on");
            gpio1.high();


             } else if (action.equals("0")) {
            System.out.println("led is off");
           gpio1.low();
           }

           } catch (Exception e) {

           System.out.print("Exception ");
           }


           PrintWriter out = response.getWriter();

           String pagehtml = "";

            // if there was a valid LED number display a message

            if ("10".contains(action)) {
            pagehtml = pagehtml + "LED number " + action + "
selected";
            }

            String pagehtmlform = "<form action="./gpio" method="POST
            <input name="action" type="text"><input type="submit"
            value="Send"></form>";

             out.println(pagehtml + pagehtmlform);

             }

             }
```

6. Your project is still missing the `Pi4J` library. To install it, simply expand the**Dependencies** node in the **Projects explorer** associated with your**iot Maven Webapp** project. Then right-click on `jsp-api-2.0.jar` and from the context menu, click **Manually install artifact**.
7. In the **Install locally** dialog box, browse for the `pi4j-core.jar` file and click **Install locally** button to install it in your project (Figure 6-13):

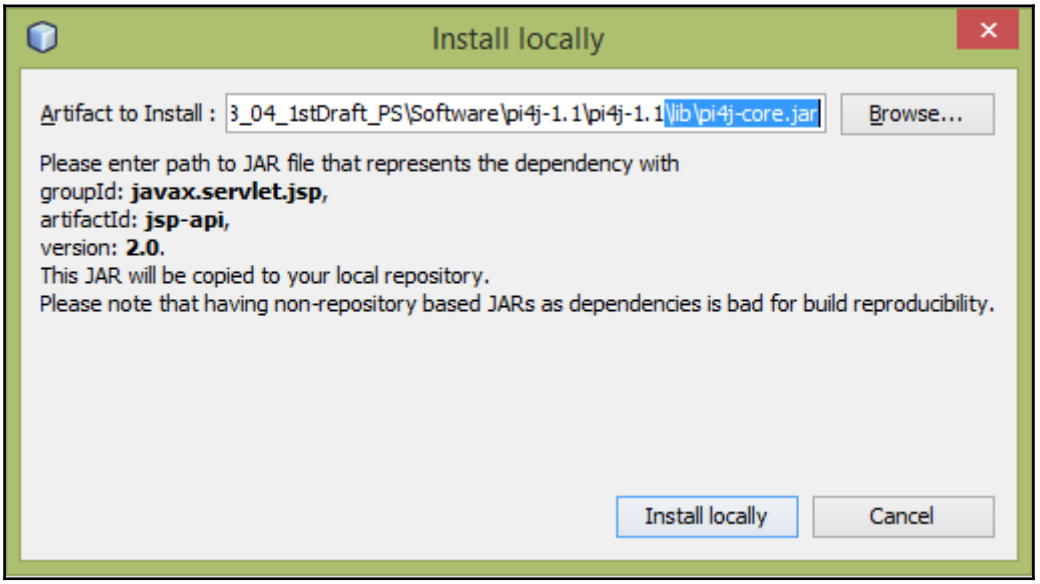


Figure 6-13: Installing an Artifact

8. The newly installed `Pi4J` library can be found by expanding the `jsp-node-2.0.jar` node, as shown in Figure 6-14:

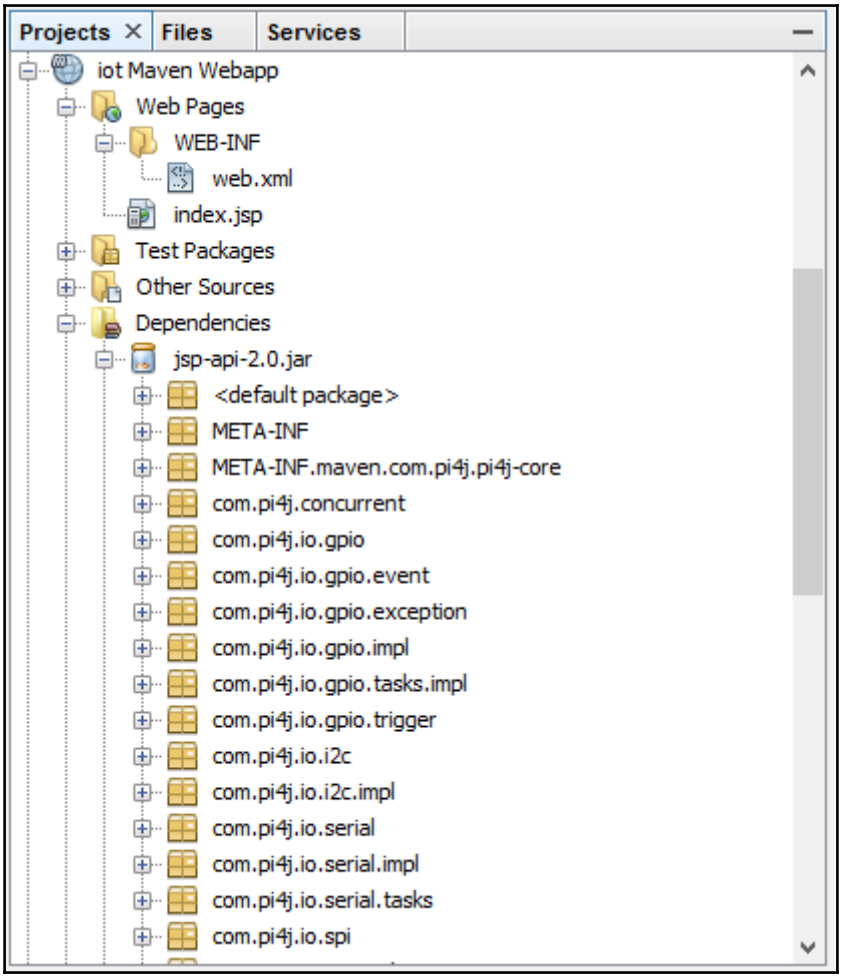


Figure 6-14: Pi4J library

9. `gpio.java` serves a HTML page that can be used to switch an LED attached to the Raspberry Pi's GPIO 1 pin on and off:

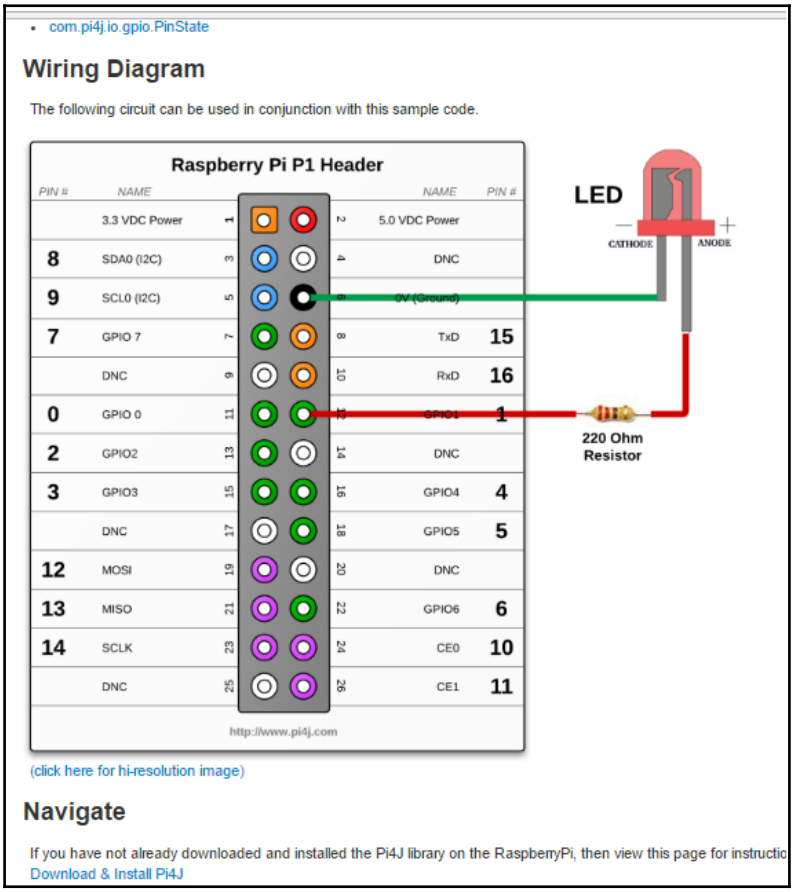


Figure 6-15: Circuit diagram - LED attached to GPIO 1. Image credits: Courtesy of `http://www.pi4j.com`

10. The server application accepts the value 1 to turn the LED on and the value 0 to turn the LED off. The value is sent to the server using the `POST` method and the URL pattern should match the name of the servlet, which is `/gpio`. The following code snippet will handle that:

```
    if ("10".contains(action)) {
pagehtml = pagehtml + "LED number " + action + "
     selected";
}
```

```
String pagehtmlform = "<form action="./gpio"
    method="POST"><input name="action"  type="text">
    <input type="submit" value="Send"></form>";

out.println(pagehtml + pagehtmlform);
```

11. The deployment of the application from a Windows computer to the Raspberry Pi is quite easy with FileZilla. Every Java-based web project generates a **Web application ARchive (WAR)** file, with an extension of `.war`. This is the distributed collection of Java server pages, Java servlets, Java classes, XML files, tag libraries, static web pages, and other resources that together constitute a web application. The `iot.war` file for this project can be found in the target directory of the `iot` project, as shown in Figure 6-16:

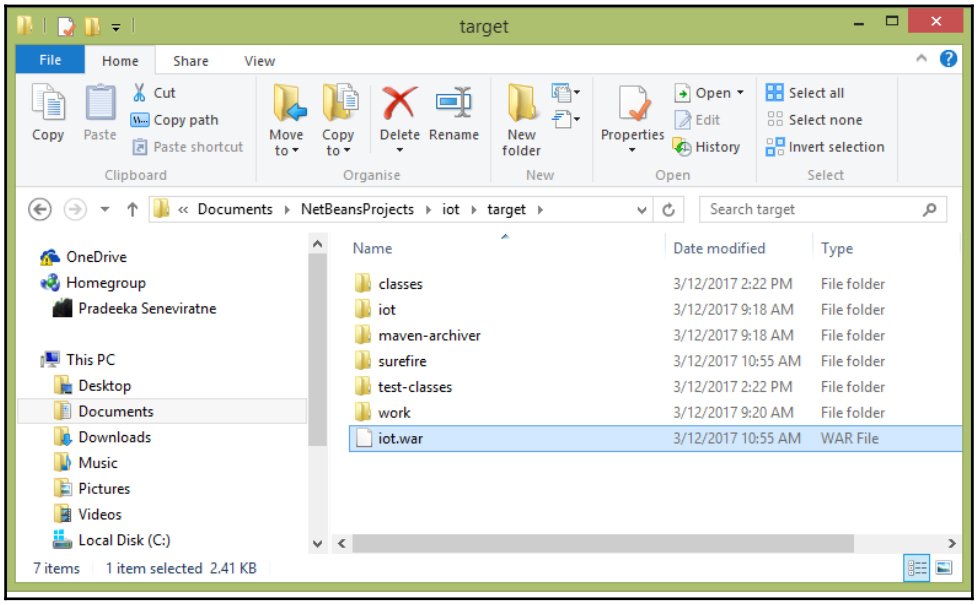


Figure 6-16: iot.war file in target folder

Copying iot.war file to the Raspberry Pi

We assume you have already installed FileZilla on your Windows computer. If not, download and install it from.

1. Start FileZilla and connect it to your Raspberry Pi with the host, which is the IP address, username, and password. This should be provided as `sftp://pi@192.168.1.2`, where we assume the IP address of the Pi is `192.168.1.2`. Change the IP address according to your Raspberry Pi.
2. Browse the `webapps` folder located in `/home/pi/jetty-distribution-9.4.2.v20170220` as shown in Figure 6-17. The `webapps` folder of the Jetty server can be used to host any web application that you would like to run:

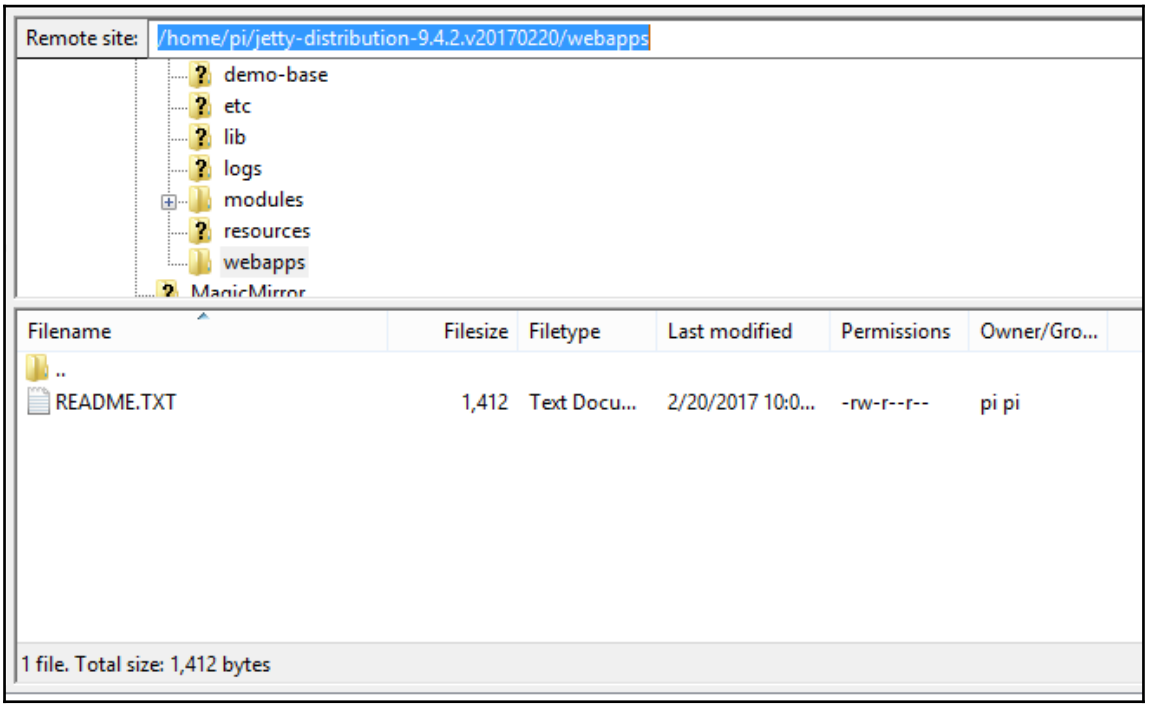


Figure 6-17:web apps directory in Jetty server

3. Simply drag the `iot.war` file from the *target* directory in **Windows explorer** to the `webapps` directory listed in FileZilla. In a few seconds, the `iot.war` file will copy to the Raspberry Pi, to the `/home/pi/jetty-distribution-9.4.2.v20170220/webapps` directory:

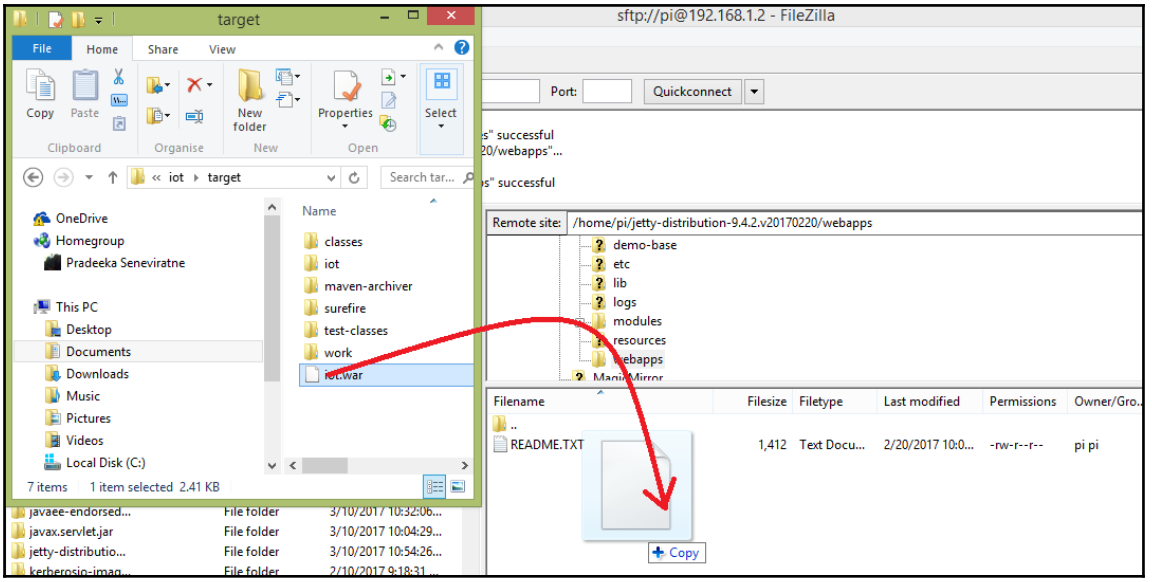


Figure 6-18: Copying iot.war to the Raspberry Pi

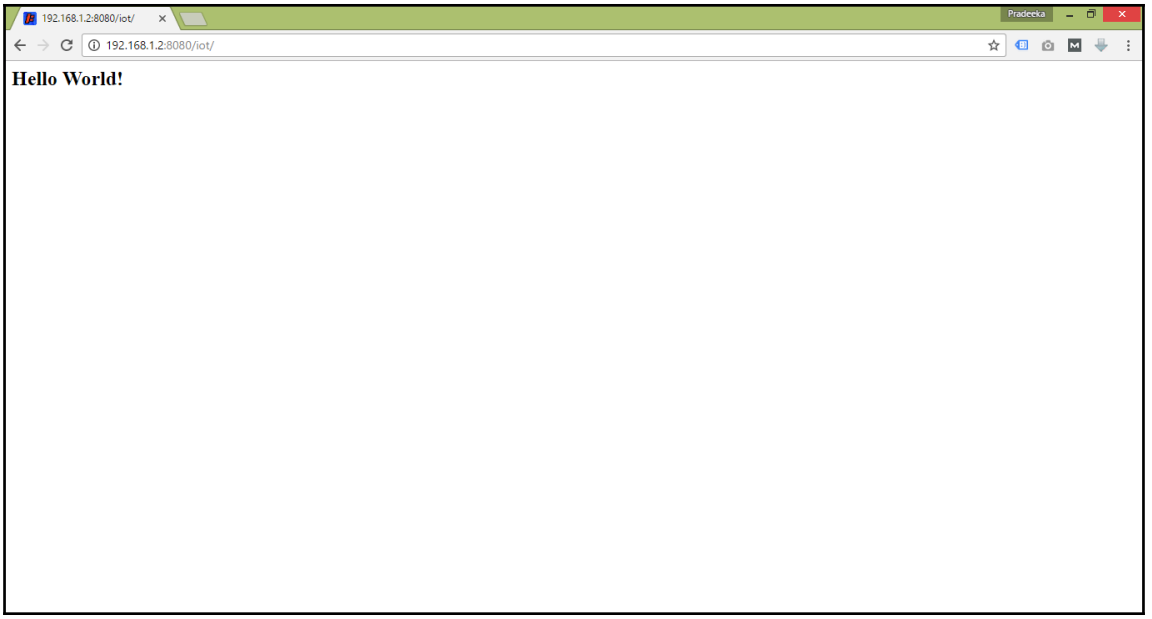
4. Now SSH to your Raspberry Pi using PuTTY and issue the following command to start the Jetty server:

```
cd jetty-distribution-9.4.2.v20170220java -jar start.jar
```

5. The server will start and you're ready to browse the IoT web application. Open your web browser and try to access the following URL, which is the entry point of your iot web application:

```
http://192.168.1.2:8080/iot/
```

6. The web browser will output a page similar to the following:



Entry point of them iot web application

7. Now let's try to access the `gpio.java` servlet, which is configured with the URL pattern `./gpio`. The URL should be:

```
http://192.168.1.2:8080/iot/gpio
```

8. The web browser will output a page similar to the following (Figure 6-19):

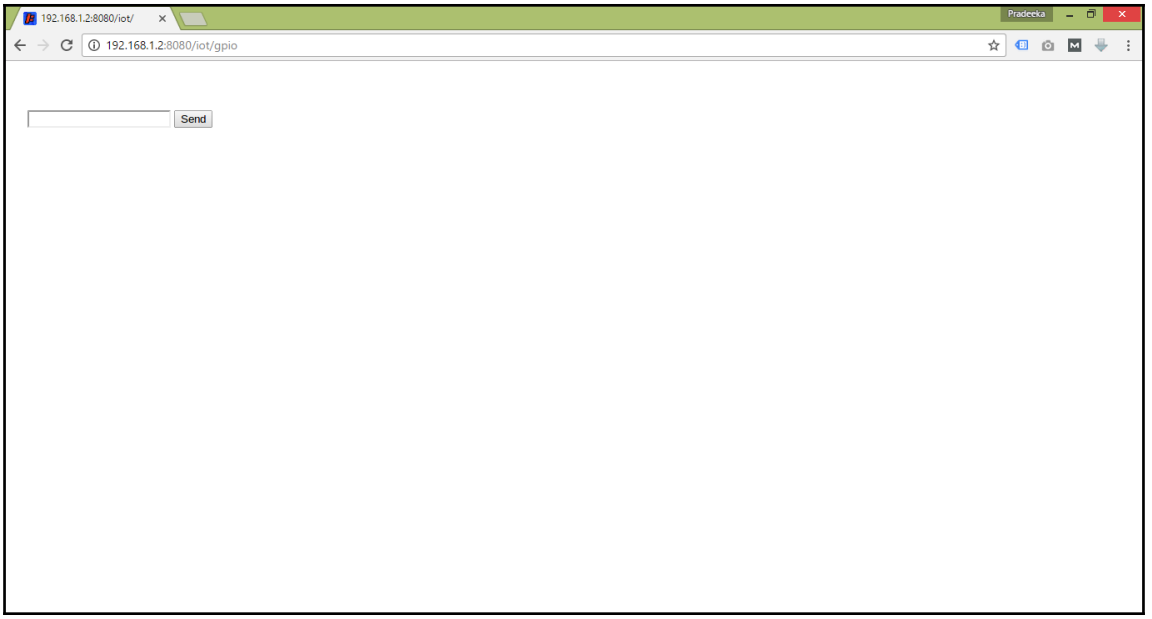


Figure 6-19: gpio servlet of the iot web application

9. You can test the application by typing the value **1** in the text box and clicking on the **Send** button. The LED attached to GPIO pin 1 will turn on. Now type `0` and click on the **Send** button. The LED will turn off.

Summary

In this chapter, you learned how to develop a Java web application with NetBeans to control a GPIO-connected LED through the network. You followed s series of steps to install and configure s Jetty server on both NetBeans and the Raspberry Pi. Finally, you deployed the web application, which is the `.war` file generated by the build process, to the Jetty server running on the Raspberry Pi.

The `Chapter 7`, *Security Camera with Face Recognition*, presents how to use OpenCV and Java with the Raspberry Pi to develop a video application with face recognition that can be used to build a security camera.

7
Security Camera with Face Recognition

In `Chapter 3`, *A Social and Personal Digital Photo Frame*, you learned how to build a multimedia application, which is a digital photo frame based on the Raspberry Pi. Now you're going to take a major step forward, by building a real-time video processing application, which is a security camera with face recognition. Face recognition is a revolutionary technology and it can be used in a wide range of applications such as:

- People tagging in photos
- Gaming
- Price comparison
- Making mental notes
- Identifying TV shows
- Augmented reality
- Image searching
- Solving sudoku puzzles
- Security

Besides providing security through camera-based applications at both day and night, by displaying videos on a screen or recording video using a hard disk, face detection can be used to identify human faces among moving objects (people, animals, vehicles, and so on) within the field of view of the camera. It's time to build a security camera with human face detection.

In this chapter, you will:

- Learn how to download and install OpenCV on Windows
- Learn how to download, build, and make OpenCV on the Raspberry Pi
- Configure VM options for OpenCV with NetBeans
- Connect the camera module with the Raspberry Pi
- Capture images with the camera module and save them to disk
- Capture video with the camera module and display it in a window based on JavaFX
- Detect human faces in video frames with Haar cascade classifiers and highlight them

To build this project, you will need the following things:

- Raspberry Pi 3 board
- Raspberry Pi camera module with mount, about £27.50 at PIMORONI (`https://shop.pimoroni.com/products/raspberry-pi-camera-module-v2-1-with-mount`)

Raspberry Pi camera module

The Raspberry Pi camera module (Figure 7-1) helps you to make still photographs and high definition videos with all models of Raspberry Pi. The latest version of the camera module is V2.1 at the time of writing this book, but all previous versions should work with this project. The camera module offers the following features: Sony IMX219 8-megapixel sensor:

- Supports 1080p30, 720p60, and VGA90 video modes
- Supports still capture

Figure 7-1: Raspberry Pi camera module V2.1. Image credits: SparkFun electronics.
https://www.flickr.com/photos/sparkfun/9367415640

Connecting the camera module to the Raspberry Pi

All Raspberry Pi models offer a CSI port to attach the camera module through a 15 cm ribbon cable. Figure 7-2 shows a camera module attached to the Raspberry Pi's CSI port:

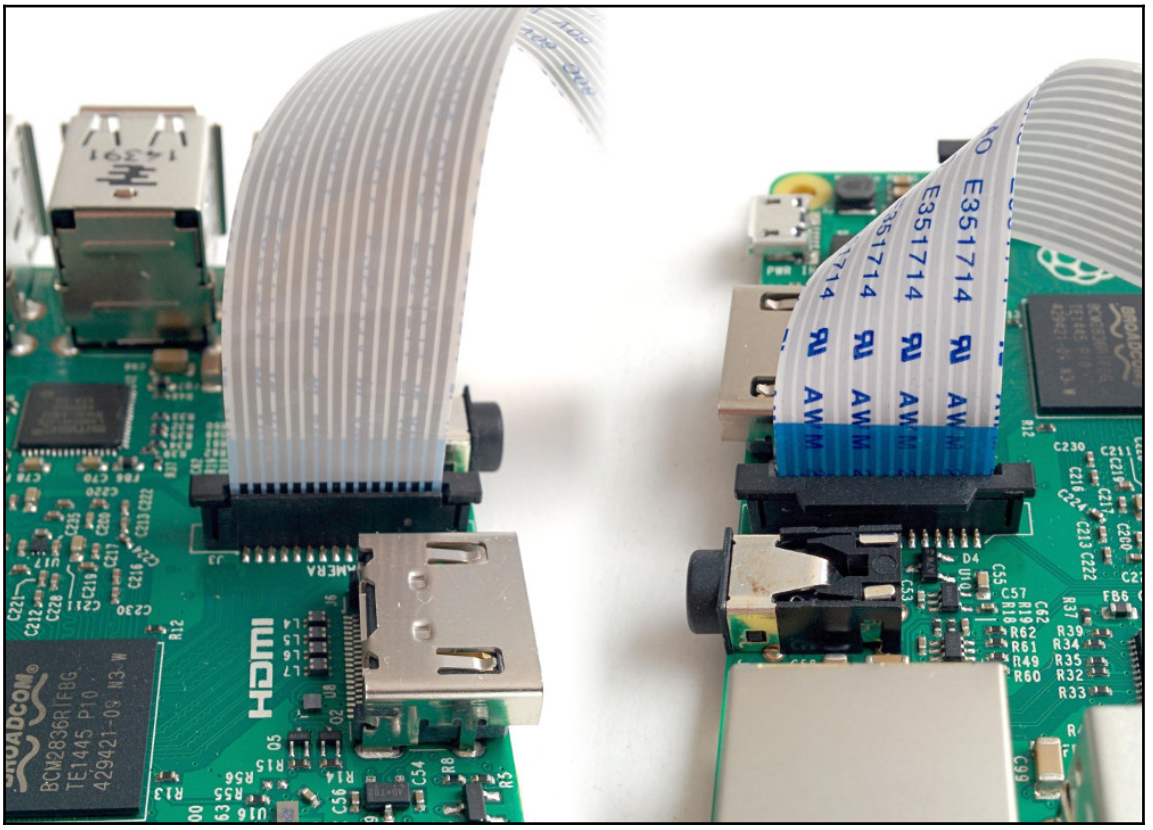


Figure 7-2: Camera module attached to the CSI port. Image credits: `https://www.raspberrypi.org`(`https://www.raspberrypi.org/learning/getting-started-with-picamera/worksheet/`)

Alternatively, you can use a USB web camera attached to the Raspberry Pi through one of the USB ports. You can find a reliable web camera brand with the model number at `http://elinux.org/RPi_USB_Webcams`.

OpenCV

The Raspberry Pi camera module can be accessed through the MMAL, V4L APIs, and some third-party libraries, including the Picamera Python library. When you need to access the camera module with Java, a well-written, feature-rich library is available, which is OpenCV.

OpenCV is currently available for Windows, Linux/Mac, Android, and iOS as pre-built libraries. However, you can build the `OpenCV` library for your preferred development environment using the source code. For Raspberry Pi, we will be building the `OpenCV` library to work with Java in a few easy steps.

Downloading and installing OpenCV on Windows

In order to write Java applications with OpenCV, first you have to download and install Open CV on your computer. The following steps explain how to do this:

1. Download the latest release of OpenCV from `http://opencv.org/downloads.html`. The page (Figure 7-3) will list all the versions released. Click on the **OpenCV for Windows** link under **VERSION 3.2.0**, dated **2016-12-23**, which is the latest release at the time of writing this book:

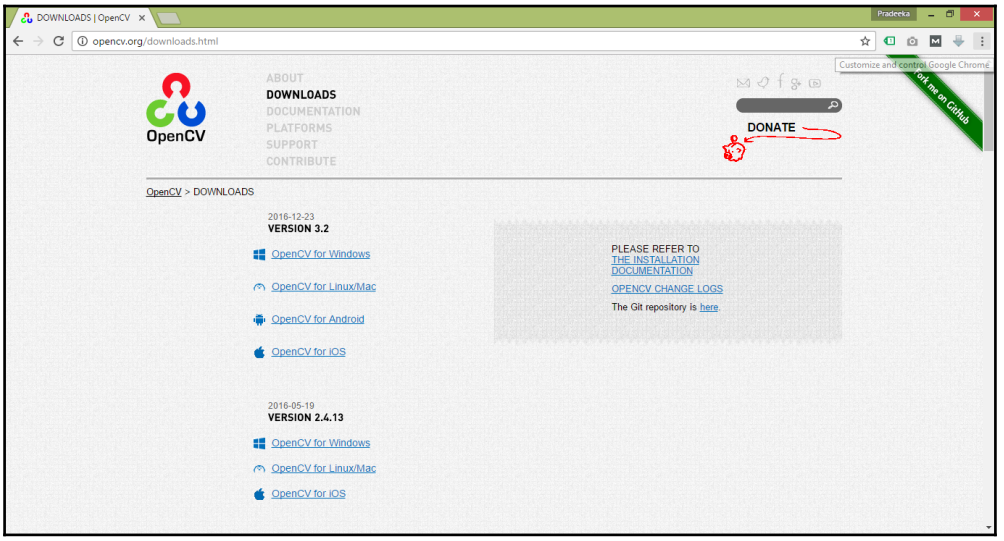


Figure 7-3: OpenCV v3.2 for Windows

2. Run the downloaded executable file named `opencv-3.2.0-vc14.exe` (or similar). The file will open with a self-extractor (Figure 7-4). Browse to the location that you want to extract the files to, and click the **Extract** button:

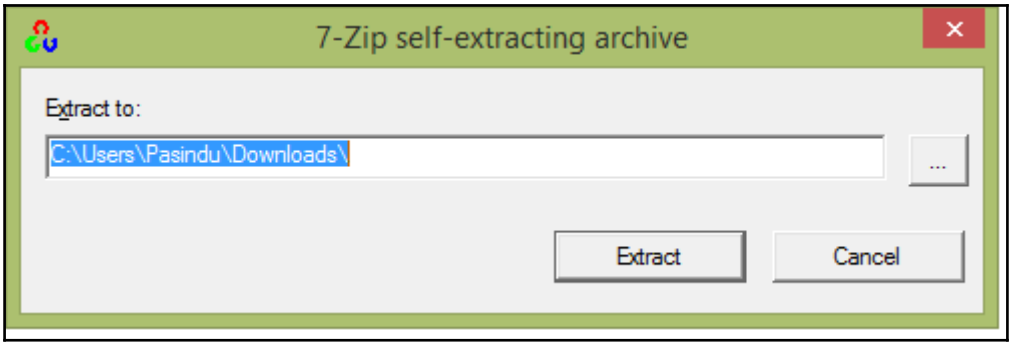


Figure 7-4: Self-extracting archive

3. After extracting the archive, you will get a folder named `opencv`.

Creating the Java project

1. Open your NetBeans IDE and create a new Java application by clicking on **File** | **New Project**.
2. In the **New Project** wizard, click **Java** under **Categories** and click **Java Application** under **Projects**. Then click the **Next** button.
3. Type `SecurityCamera` for the **Project Name** and `com.packt.B05688.chapter7.SecurityCamera` for create `Main` class. Click the **Finish** button to create the project.

Adding the OpenCV library to your Java project

Expand the security camera project node and right-click on the `Library` folder. From the context menu, click on **Add Library**.

1. In the **Add Library** dialog box, click the **Create** button (Figure 7-5):

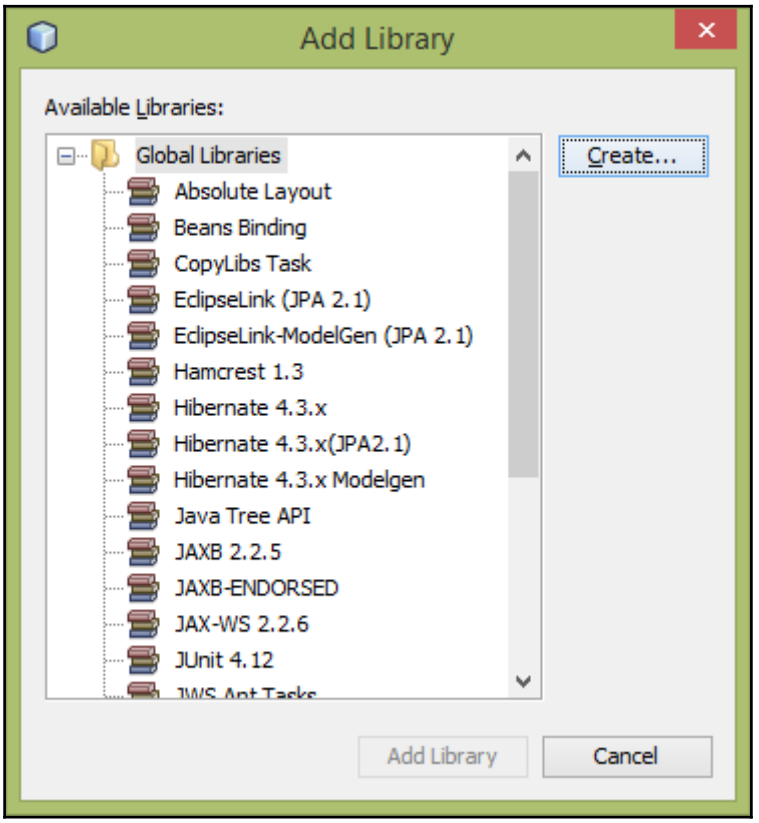


Figure 7-5: Add Library

2. In the **Create New Library** dialog box (Figure 7-6), type `OpenCV3Library` for the **Library Name** and click the **OK** button:

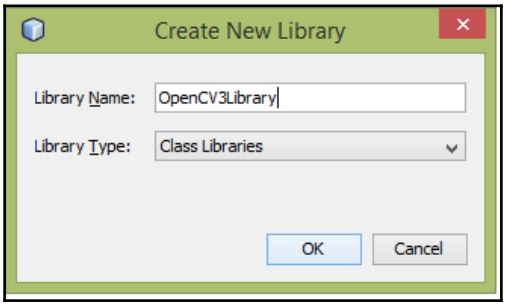


Figure 7-6: Create New Library

3. In the Customize Library dialog box (Figure 7-7), click the **Add JAR/Folder...** button:

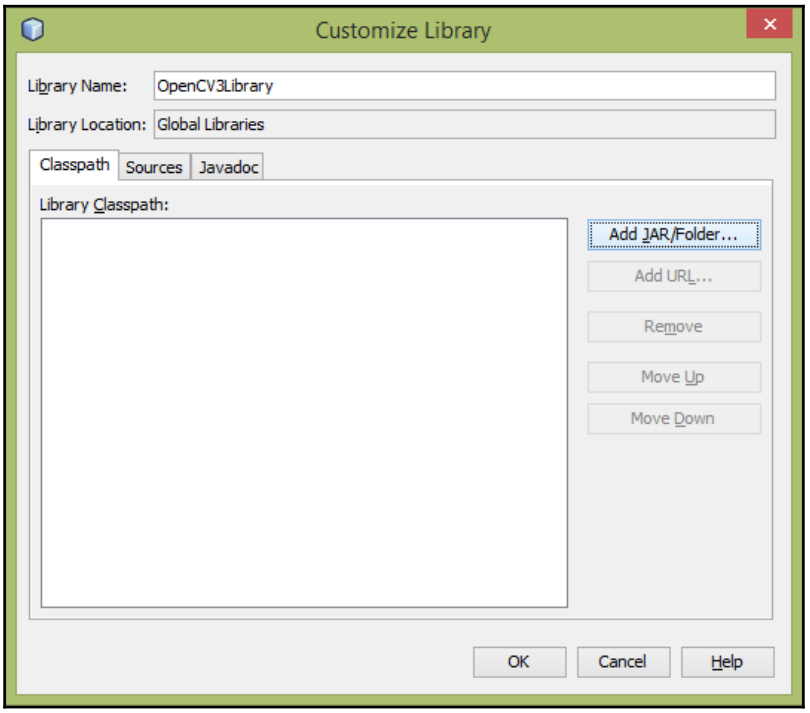


Figure 7-7: Customize Library

4. Browse the previously extracted `opencv` folder and select the `opencv-320.jar` file located in `your-drive:/opencv/build/java/`. Then click the **Add Jar/Folder** button (Figure 7-8):

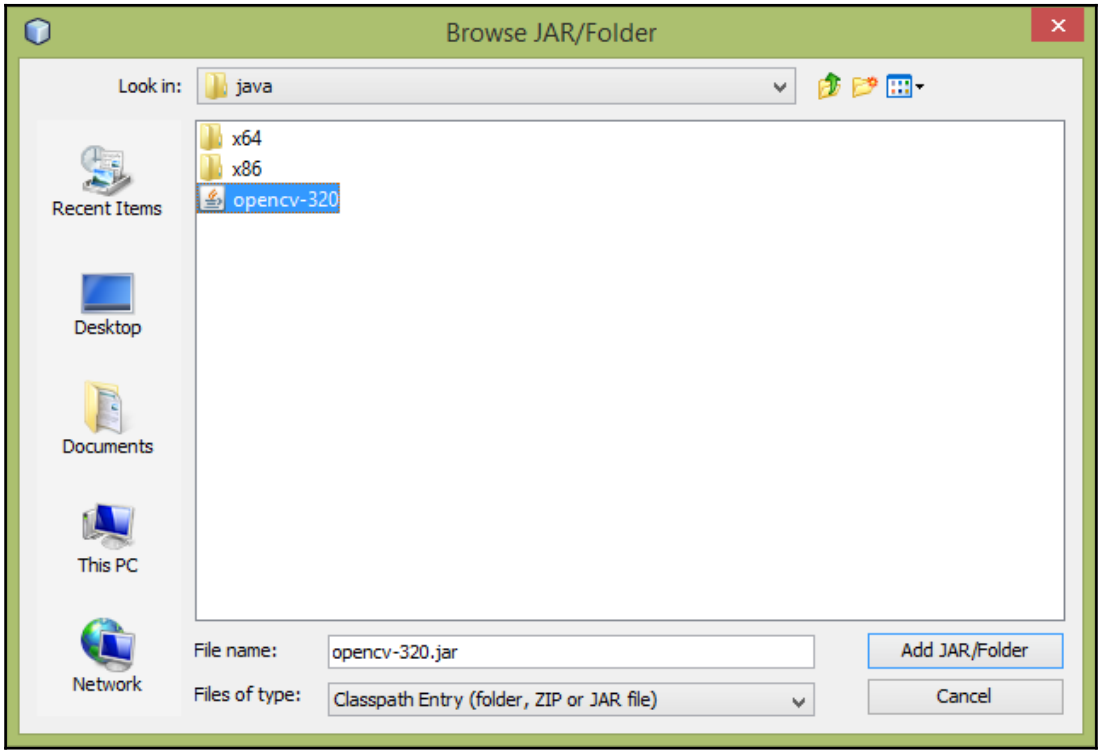


Figure 7-8: Browse JAR/Folder

5. The library path will be added to the **Library Classpath**, as shown in Figure 7-9. Click **OK** to add the library to the available libraries:

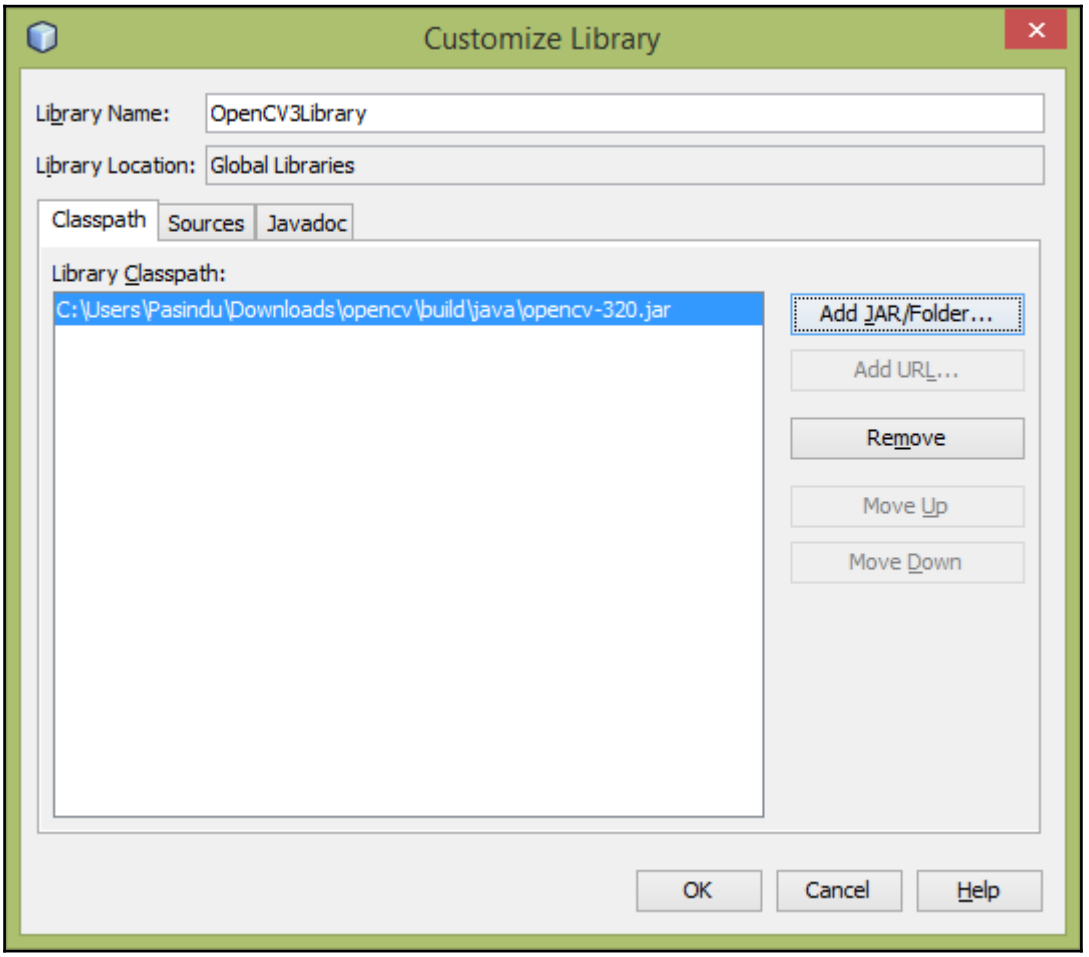


Figure 7-9: Customize Library

6. In the **Add Library** dialog box (Figure 7-10), click the **Add Library** button to add the **OpenCV3Library** to your Java project:

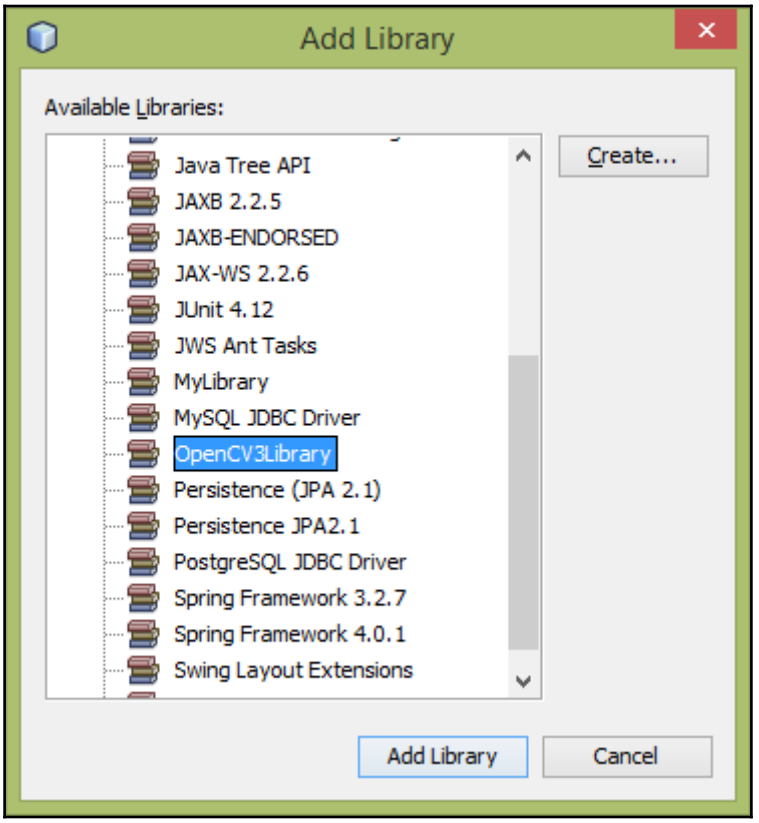


Figure 7-10: Add Library

7. `OpenCV3Library` is now visible as `OpenCV3Library - opencv-320.jar` under the `Libraries` node in the **Projects** explorer (Figure 7-11):

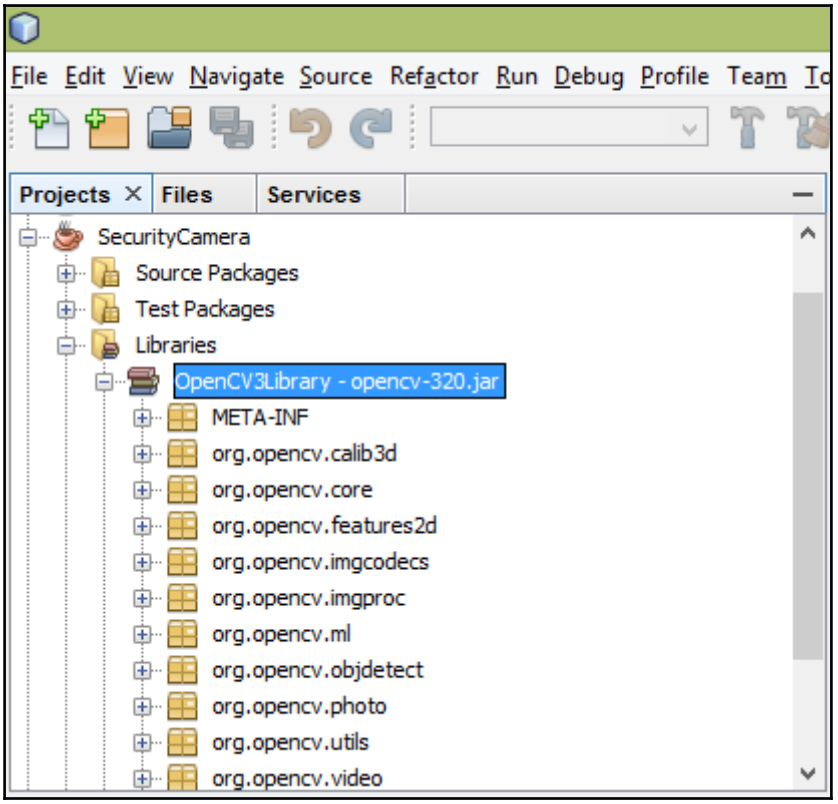


Figure 7-11: NetBeans Project Explorer

8. Now, right-click on the **SecurityCamera** node in the project explorer and from the context menu, click **Properties**.

9. In the **Project Properties** dialog box, under **Categories**, click the **Run** node. Type the following in the **VM Options** text box (Figure 7-12):

```
-Djava.library.path=C:\Users\Pasindu\Downloads\opencv\build\java\x64
```

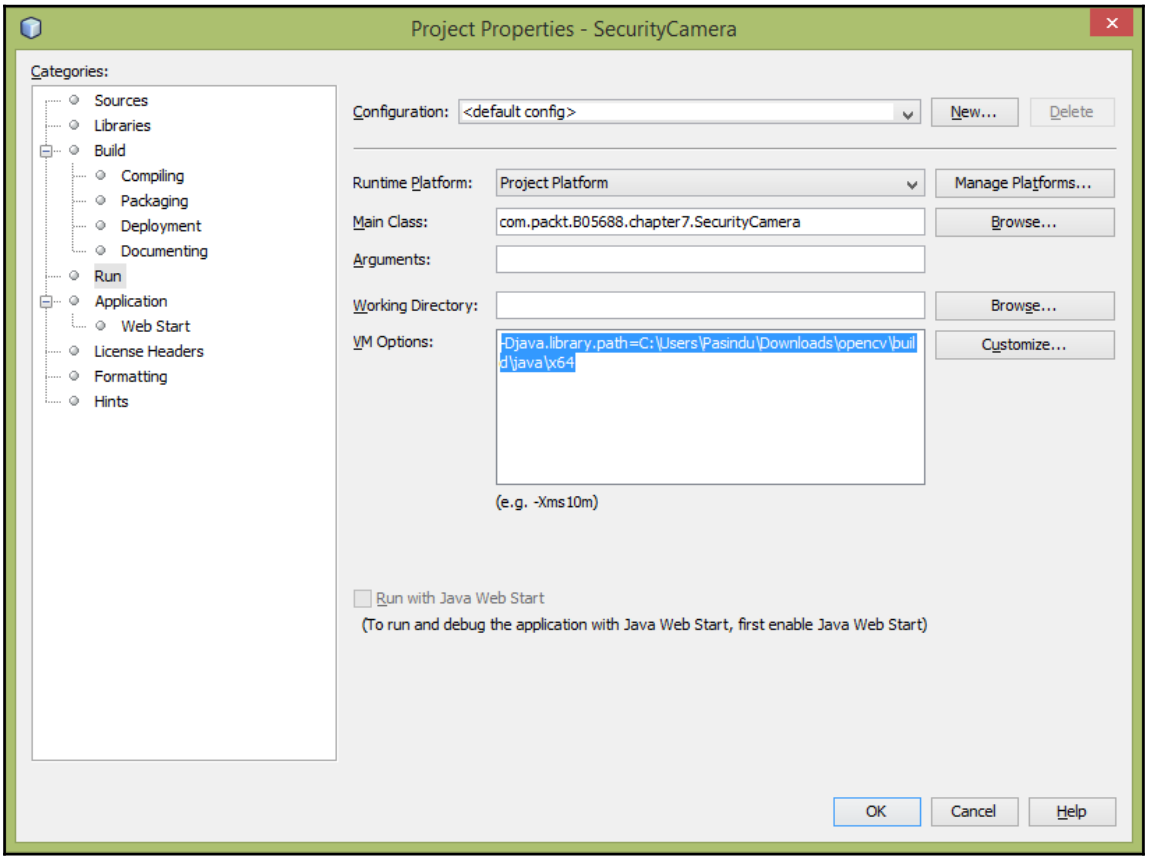


Figure 7-12: Project Properties - Security Camera

This will point to the `opencv_java320.dll` file in the `/opencv/build/java/x64` folder. Change `x64` to `x86` if you are running 32-bit Windows. Click the **OK** button to complete the configuration:

1. Now you have successfully created your development environment to work with OpenCV Java projects. Let's start with a simple Java program. In the Project explorer, open the `SerurityCamera.java` file and type the code shown in Listing 7-1:

```
package com.packt.B05688.chapter7;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;

public class SecurityCamera {

 public static void main(String[] args) {
 System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
 Mat mat = Mat.eye(3, 3, CvType.CV_8UC1);
 System.out.println("mat = " + mat.dump());
 }
}
```

2. Run the project and you will get an output similar to Figure 7-13 in the output window:

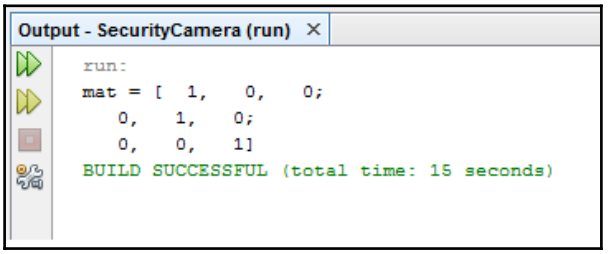


Figure 7-13: Output in Windows environment

This ensures that OpenCV is correctly configured with Java in a Windows environment. However, we need to run standalone OpenCV projects on the Raspberry Pi without the aid of a Windows OS installed computer. The next section will explain how to install OpenCV on a Raspberry Pi and configure it with Java.

Downloading and building OpenCV on Raspberry Pi

Building OpenCV from the source is the easiest way to grab OpenCV for your Raspberry Pi Java projects. The following steps will guide you through how to download, build and make OpenCV on the Raspberry Pi. Connect it to your Raspberry Pi through SSH using PuTTY:

1. Install the following build essentials on the Raspberry Pi to have all the necessary tools to build the OpenCV source:

```
sudo apt-get -y install ant build-essential cmake
cmake-curses-gui
```

2. The GitHub page (Figure 7-14) at `https://github.com/opencv/opencv/releases` lists all the releases of OpenCV and the latest release is **OpenCV 3.2.0**, dated Dec 23, 2016 at the time of writing this book. We need only the source code of **OpenCV 3.2.0**, labeled as **Source code (zip)** to build OpenCV on the Raspberry Pi:

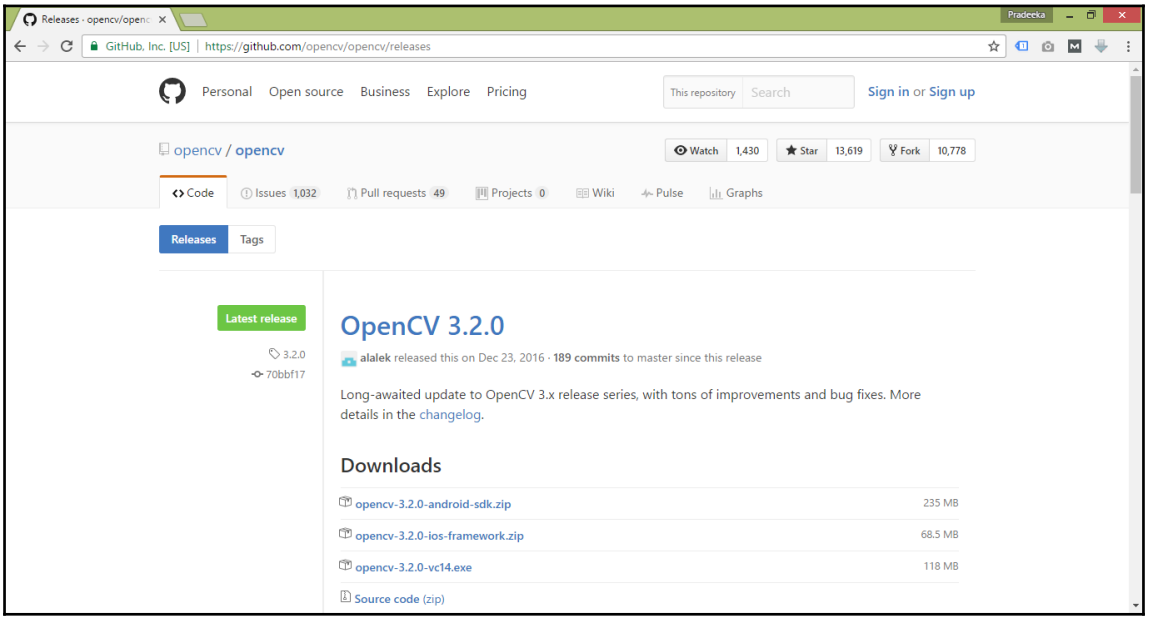


Figure 7-14: OpenCV 3.2.0 release downloads

You can download the OpenCV source (`3.2.0.zip`) using the `wget` command and then unzip the `3.2.0. zip` file using the `unzip` command.

Make sure to issue this command from your `home` directory, which is `/home/pi`, by quickly navigating with the `cd ~` command. The `cd ~` command simply takes you back to your `home` directory.

```
cd ~
wget          https://github.com/opencv/opencv/archive/3.2.0.zip
unzip 3.1.0.zip
```

3. After extracting the `.zip` file, change the directory to `opencv-3.2.0` and create a new directory named `build`, then change your directory to build:

```
cd ~/opencv-3.2.0
mkdir build
cd build
```

4. Execute following command to build the source:

```
 cmake -D ANT_EXECUTABLE=/usr/bin/ant \
-D BUILD_PERF_TESTS=OFF \
-D BUILD_SHARED_LIBRARY=OFF \
-D BUILD_TESTS=OFF \
-D BUILD_opencv_python3=OFF \
-D CMAKE_BUILD_TYPE=RELEASE \
-D JAVA_AWT_INCLUDE_PATH=/usr/lib/jvm/jdk-8-oracle-
        arm32-vfp-hflt/include \
-D JAVA_AWT_LIBRARY=/usr/lib/jvm/jdk-8-oracle-arm32-
         vfp-hflt/include/jawt.h \
-D JAVA_INCLUDE_PATH=/usr/lib/jvm/jdk-8-oracle-arm32-
        vfp-hflt/include \
-D JAVA_INCLUDE_PATH2=/usr/lib/jvm/jdk-8-oracle-arm32-
        vfp-hflt/include/linux \
-D JAVA_JVM_LIBRARY=/usr/lib/jvm/jdk-8-oracle-arm32-
        vfp-hflt/include/jni.h \
-D WITH_JPEG=OFF ..
```

After building the source, make the build by running the following command. The process will take about 30 minutes to complete on a Raspberry Pi 3:

```
make -j4
```

1. A file named `opencv-320.jar` will be created under the `/home/pi/opencv-3.1.0/build/bin` directory.
2. Right-click on the `SecurityCamera` node in the *project explorer*, then from the context menu, click **Properties**. In the **Project Properties** dialog box of the `SecurityCamera` project, click the **Run** node (Figure 7-15):

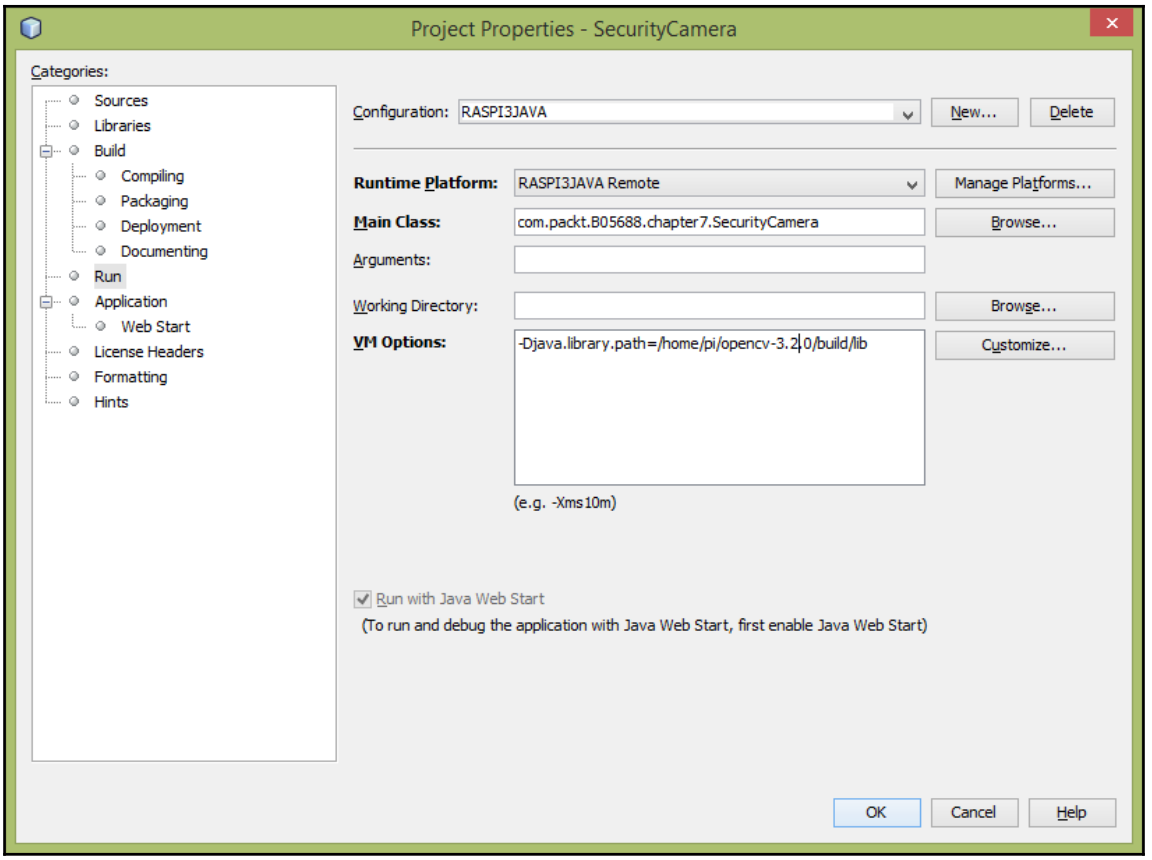


Figure 7-15: Project Properties - Security Camera

3. Then, select the Configuration as `RASPI3JAVA`, which has already been created in `Chapter 1`, *Setting up Your Raspberry Pi*. Now, type the following in the **VM Options** text box and click the **OK** button to save the settings:

```
-Djava.library.path=/home/pi/opencv-
3.2.0/build/lib
```

4. Targeted at the Raspberry Pi. You will get the same output as previously, when we ran it on Windows:
5. Run the previous Java program (Listing 7-16) with the new settings, which are targeted at the Raspberry Pi. You will get the same output as previously, when we ran it on Windows:

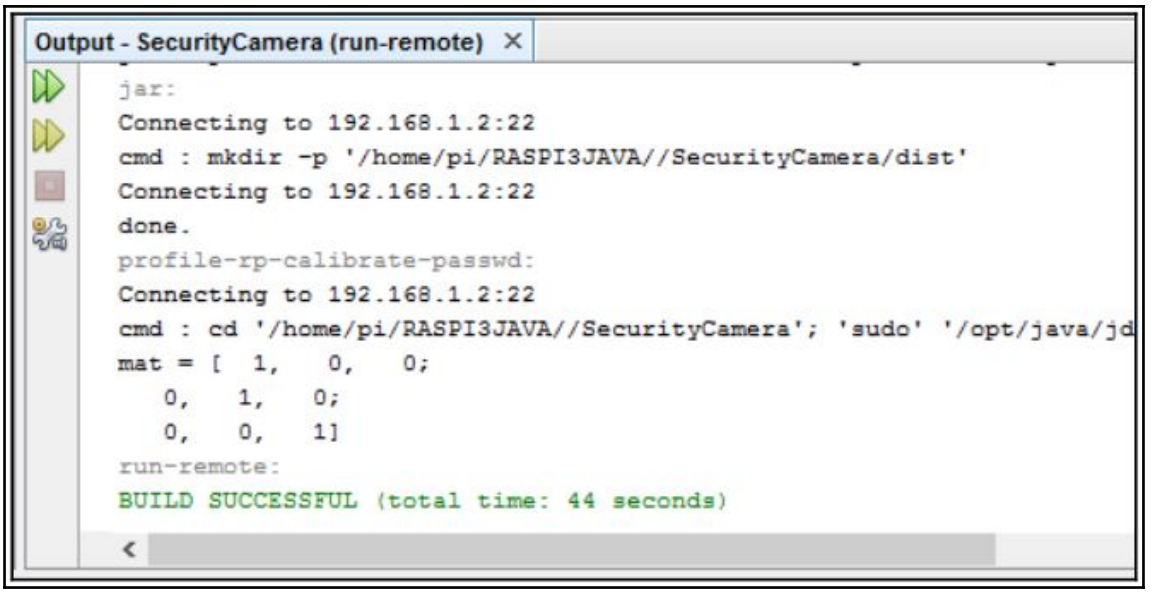


Figure 7-16: Output in Raspberry Pi (run-remote)

Working with video

as the key source for security purposes, rather than the still images. The video quality extends to full HD, which is 1080p, at a frame rate of 30.

It's time to work with video capture, which can be used as the key source for security purposes, rather than the still images. The video quality extends to full HD, which is 1080p, at a frame rate of 30. Listing 7-2 shows a Java program that can be used to capture video using the Raspberry Pi camera module and save a snapshot to the SD card. Replace Listing 7-1 with this:

Listing 7-2: `SecurityCamera.java`

```
package com.packt.B05688.chapter7;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.videoio.VideoCapture;

public class SecurityCamera {

 static {
 System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
 }

 public static void main(String[] args) {
 Core.setErrorVerbosity(false);
 VideoCapture camera = new VideoCapture(0);
 delay(1000);

 if(!camera.isOpened()){
 System.out.println("Camera Error");
 }
 else {
 System.out.println("Camera OK");
 }

 Mat frame = new Mat();
 camera.read(frame);

Imgcodecs.imwrite("/home/pi/RASPI3JAVA/SecurityCamera/captured/capt
ure1.png", frame);
 }

 private static void delay(long ms) {
 try {
 Thread.sleep(ms);
 }
 catch (InterruptedException e) {
 System.out.println(e);
 }
```

```
    }

}
```

Let's take a look at some of the important sections in this Java program, so you can modify it to suit your requirements:

1. The `VideoCapture` class has all the functionalities required for video manipulation. The `camera` is the object of the `VideoCapture` class:

```
VideoCapture camera = new VideoCapture(0);
```

   The video is composed of a set of images, called frames. In videography, the frame rate specifies the number of images that can be captured per second.

2. The `isOpened` method can be used to check whether the binding of the class to a video source was successful or not:

```
if(!camera.isOpened()){
System.out.println("Camera Error");
}
else {
System.out.println("Camera OK");
}
```

3. The `read` method grabs, decodes, and returns the next video frame:

```
camera.read(frame);
```

4. The `imwrite` method of the `Imgcodecs` class writes the video frame as an image file to disk:

```
Imgcodecs.imwrite("/home/pi/RASPI3JAVA/SecurityCamera/captured/capt
ure1.png", frame);
```

5. Now, rebuild the `SecurityCamera` project with NetBeans and run it from the Raspberry Pi using the following command:

```
sudo java –jar
        /home/pi/RASPI3JAVA/SecurityCamera/dist/SecurityCamera.jar
```

6. Open the *captured* directory to see the `capture1.png` image. Each time you run the project, the `capture1.png` image file overwrites with the new video frame captured by the `read` method.

We can improve this program to show real-time video in a window using JavaFX. Listing 7-3 presents a Java program that can be used to display the video captured by the camera module connected to the Raspberry Pi:

Listing 7-3: `SecurityCamera.java`

```
package com.packt.B05688.chapter7;

import java.io.ByteArrayInputStream;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfByte;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.videoio.VideoCapture;
import org.opencv.videoio.Videoio;

public class SecurityCamera extends Application {

 static {
 System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
 }

 private boolean isStart = false;
 private VideoCapture capture;
 private ScheduledExecutorService timer;

 private BorderPane root;
 private VBox vboxCenter;
```

```
private ImageView frame;
private HBox hboxBottom;
private Button videoButton, exitButton;

@Override
public void start(Stage primaryStage) {

initGui();

capture = new VideoCapture();

exitButton.setOnAction((ActionEvent event) -> {
System.exit(0);
});

videoButton.setOnAction((ActionEvent event) -> {

if (!isStart) {
frame.setFitWidth(640);
frame.setFitHeight(480);
frame.setPreserveRatio(true);

capture.open(0);
capture.set(Videoio.CAP_PROP_FRAME_WIDTH, 640);
capture.set(Videoio.CAP_PROP_FRAME_HEIGHT, 480);

if (capture.isOpened()) {
isStart = true;

Runnable frameGrabber = new Runnable() {

@Override
public void run() {
Image imageToShow = grabFrame();
frame.setImage(imageToShow);
}
};

timer = Executors.newSingleThreadScheduledExecutor();
timer.scheduleAtFixedRate(frameGrabber,
0, 33, TimeUnit.MILLISECONDS);

    videoButton.setText("Stop");
}
else {
System.err.println("Open camera error!");
}
}
```

```
else {
isStart = false;
videoButton.setText("Start");

    try {
    timer.shutdown();
timer.awaitTermination(33, TimeUnit.MILLISECONDS);
}
catch (InterruptedException e) {
System.err.println(e);
}

capture.release();
frame.setImage(null);
}
});

Scene scene = new Scene(root, 800, 640);
primaryStage.setTitle("Pixel Demo 01");
primaryStage.setScene(scene);
primaryStage.show();
}

private Image grabFrame() {
Image result = null;
Mat image = new Mat();

if (capture.isOpened()) {
    capture.read(image);

if (!image.empty()) {
result = mat2Image(".png", image);
}
}

return result;
}

public static Image mat2Image(String ext, Mat image) {
MatOfByte buffer = new MatOfByte();
Imgcodecs.imencode(ext, image, buffer);
return new Image(new
        ByteArrayInputStream(buffer.toArray()));
}

private void initGui() {
root = new BorderPane();
```

```
vboxCenter = new VBox();
vboxCenter.setAlignment(Pos.CENTER);
vboxCenter.setPadding(new Insets(5, 5, 5, 5));
     frame = new ImageView();
vboxCenter.getChildren().addAll(frame);
root.setCenter(vboxCenter);

hboxBottom = new HBox();
hboxBottom.setAlignment(Pos.CENTER);
hboxBottom.setPadding(new Insets(5, 5, 5, 5));
videoButton = new Button("Start");
exitButton = new Button("Exit");
hboxBottom.getChildren().addAll(videoButton,
        exitButton);
root.setBottom(hboxBottom);
}

public static void main(String[] args) {
launch(args);
}

}
```

Rebuild the `SecurityCamera` project with NetBeans and run it from the Raspberry Pi using the following command:

```
sudo java -jar /home/pi/RASPI3JAVA/SecurityCamera/dist/SecurityCamera.jar
```

You will get a JavaFX-based window similar to the image shown in Figure 7-17:

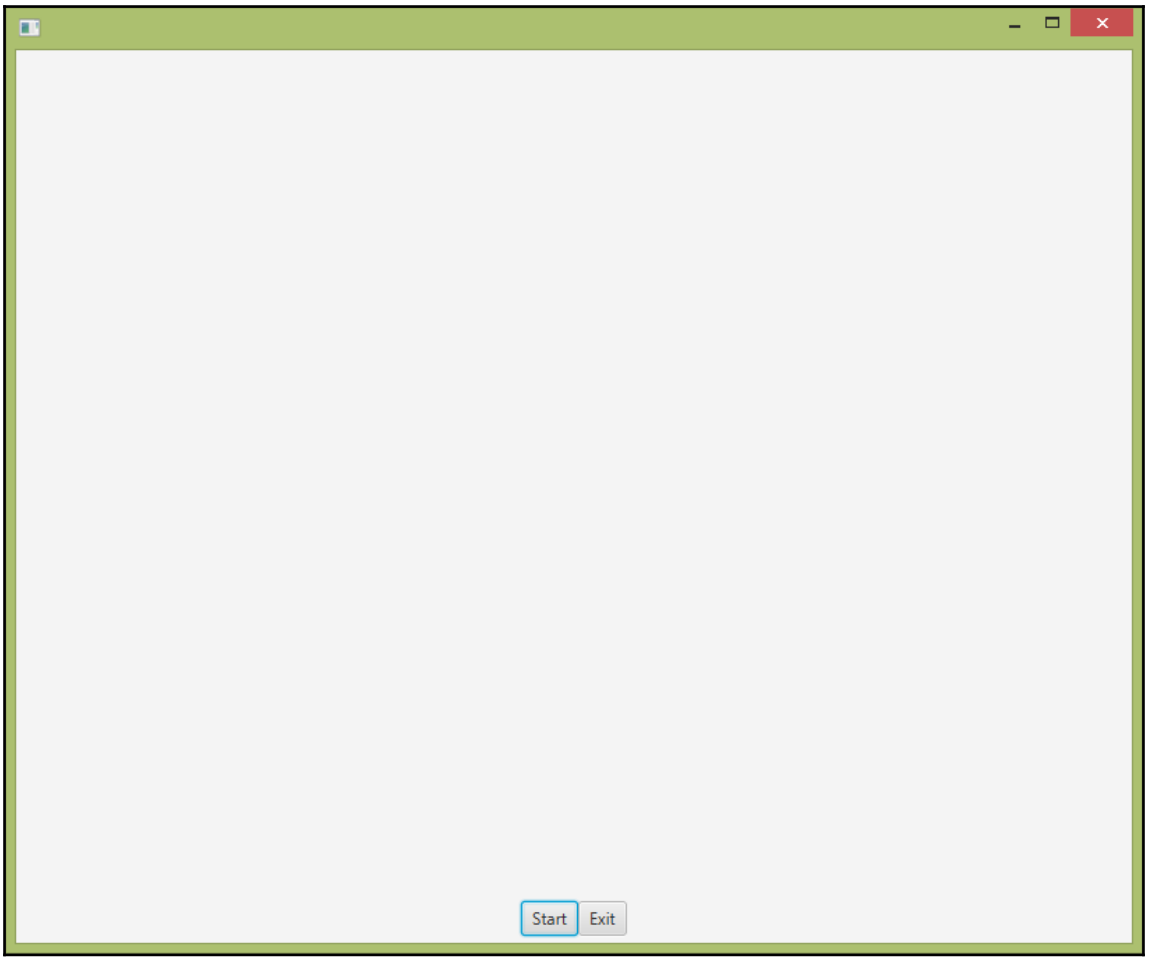


Figure 7-17: JavaFX window

7. Click the **Start** button to grab the video and the Stop button to stop the video. The sequence of grabbed images (frames) at 30 frames per second makes a real-time video similar to what is shown in Figure 7-18:

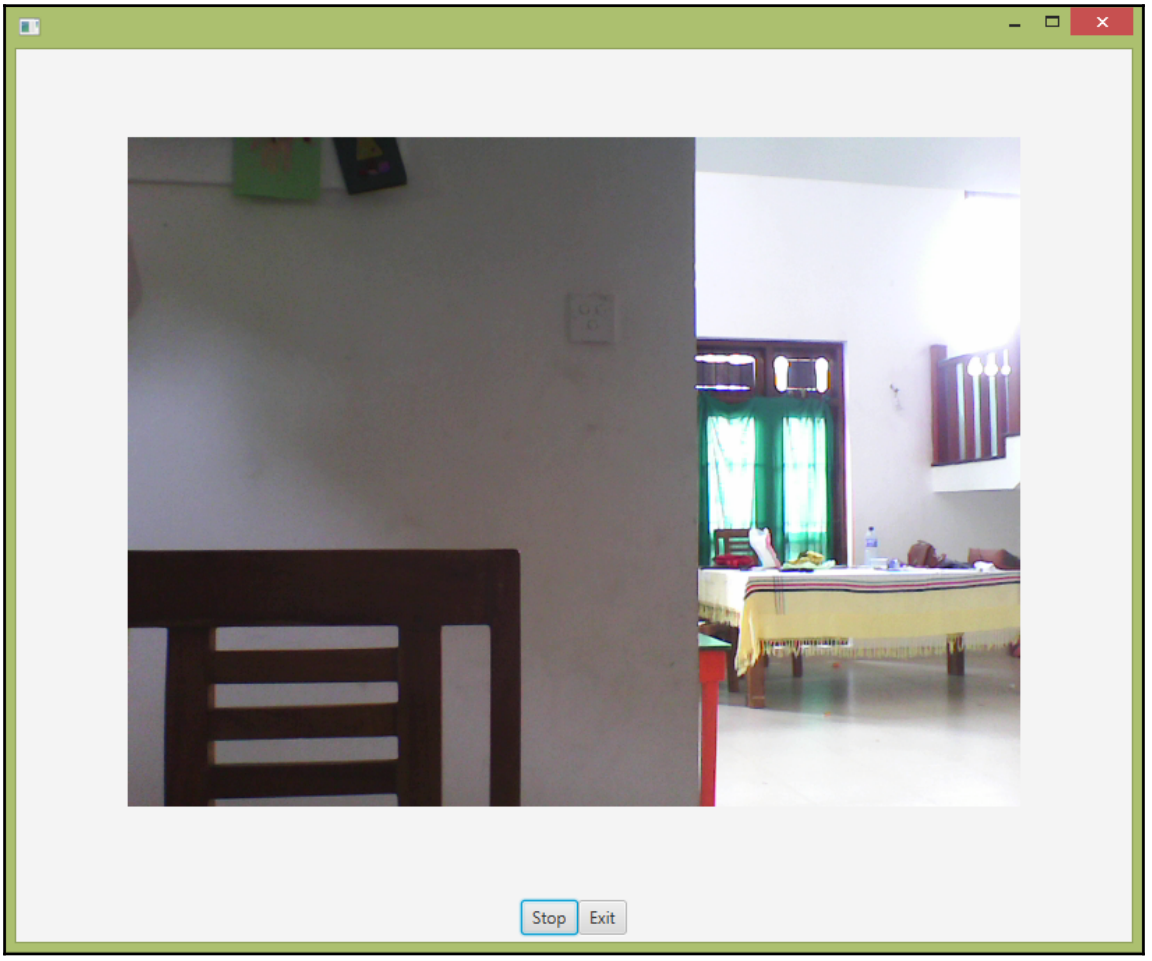


Figure 7-18: Video output

Facial recognition

Face recognition is another improvement that can be used to highlight the human face in a video output. This can be further improved to identify the correct person by matching the human face with a pre-built database.

In this chapter, we are going to use Haar-like features to encode the contrasts highlighted by the human face and its spatial relations with other objects present in the picture. Usually, these features are extracted using a cascade classifier, which has to be trained in order to recognize different objects with precision. A pre-trained classifier for face recognition can be found at `https://github.com/opencv/opencv/tree/master/data/haarcascades` in the OpenCV GitHub repository as listed following:

- `haarcascade_frontalcatface.xml`
- `haarcascade_frontalcatface_extended.xml`
- `haarcascade_frontalface_alt.xml`
- `haarcascade_frontalface_alt2.xml`
- `haarcascade_frontalface_alt_tree.xml`
- `haarcascade_frontalface_default.xml`

The Haar classifier named `haarcascade_frontalcatface.xml` is great for detecting and tracking human faces in video output.

You can copy the downloaded `haarcascade_frontalcatface.xml` file to the `/home/pi/RASPI3JAVA/SecurityCamera/` directory to use for face detection which is the front part of the human face.

Listing 7-4 presents the modified `SecurityCamera.java` program, which can be used to detect and track human faces in video output. The application can also detect multiple faces and highlight each face with a rectangle:

Listing 7-4: `SecurityCamera.java`

```
package com.packt.B05688.chapter7;

import java.io.ByteArrayInputStream;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
```

```
import javafx.stage.Stage;
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfByte;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;
import org.opencv.videoio.VideoCapture;
import org.opencv.videoio.Videoio;

import org.opencv.core.MatOfRect;


import org.opencv.core.Point;


import org.opencv.core.Rect;


import org.opencv.core.Scalar;


import org.opencv.objdetect.CascadeClassifier;


import org.opencv.objdetect.Objdetect;


public class SecurityCamera extends Application {

static {
 System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
 }

private boolean isStart = false;
 private VideoCapture capture;
 private ScheduledExecutorService timer;
 private BorderPane root;
 private VBox vboxCenter;
 private ImageView frame;
 private HBox hboxBottom;
 private Button videoButton, exitButton;



CascadeClassifier faceDetector = new
      CascadeClassifier();
```

```
@Override
 public void start(Stage primaryStage) {

initGui();

capture = new VideoCapture();

exitButton.setOnAction((ActionEvent event) -> {
 System.exit(0);
 });

videoButton.setOnAction((ActionEvent event) -> {

if (!isStart) {
 frame.setFitWidth(640);
 frame.setFitHeight(480);
 frame.setPreserveRatio(true);
capture.open(0);
 capture.set(Videoio.CAP_PROP_FRAME_WIDTH, 640);
 capture.set(Videoio.CAP_PROP_FRAME_HEIGHT, 480);

if (capture.isOpened()) {
 isStart = true;

Runnable frameGrabber = new Runnable() {

@Override
 public void run() {
 Image imageToShow = grabFrame();
 frame.setImage(imageToShow);
 }
 };

timer = Executors.newSingleThreadScheduledExecutor();
 timer.scheduleAtFixedRate(frameGrabber,
 0, 33, TimeUnit.MILLISECONDS);

videoButton.setText("Stop");
 } else {
 System.err.println("Open camera error!");
 }
 } else {
 isStart = false;
 videoButton.setText("Start");

try {
 timer.shutdown();
 timer.awaitTermination(33, TimeUnit.MILLISECONDS);
```

```
 } catch (InterruptedException e) {
 System.err.println(e);
 }

capture.release();
 frame.setImage(null);
 }
 });

Scene scene = new Scene(root, 800, 640);
 primaryStage.setScene(scene);
 primaryStage.show();
 }

private Image grabFrame() {

 faceDetector.load("haarcascade_frontalface_alt.xml");

 Image result = null;
 Mat image = new Mat();


MatOfRect faceDetections = new MatOfRect();


if (capture.isOpened()) {
 capture.read(image);

if (!image.empty()) {


if (faceDetector.empty()) {


 System.out.println("error in face detector");


 }


 faceDetector.detectMultiScale(image, faceDetections);


 System.out.println(String.format("Detected %s faces",
faceDetections.toArray().length));
```

```
for (Rect rect : faceDetections.toArray()) {


 Imgproc.rectangle(image, new Point(rect.x, rect.y),
      new Point(rect.x + rect.width, rect.y +
      rect.height),


 new Scalar(0, 255, 0), 2);


}


result = mat2Image(".png", image);
 }
 }

return result;
 }

public static Image mat2Image(String ext, Mat image) {
 MatOfByte buffer = new MatOfByte();
 Imgcodecs.imencode(ext, image, buffer);
 return new Image(new ByteArrayInputStream(buffer.toArray()));
 }

private void initGui() {
 root = new BorderPane();

vboxCenter = new VBox();
 vboxCenter.setAlignment(Pos.CENTER);
 vboxCenter.setPadding(new Insets(5, 5, 5, 5));
 frame = new ImageView();
 vboxCenter.getChildren().addAll(frame);
 root.setCenter(vboxCenter);

hboxBottom = new HBox();
 hboxBottom.setAlignment(Pos.CENTER);
 hboxBottom.setPadding(new Insets(5, 5, 5, 5));
 videoButton = new Button("Start");
 exitButton = new Button("Exit");
 hboxBottom.getChildren().addAll(videoButton, exitButton);
 root.setBottom(hboxBottom);
}

public static void main(String[] args) {
 launch(args);
```

```
}
}
```

Let's look at the most important sections of the modified `SecurityCamera.java` program:

1. Import the following additional classes to the program to use face detection and make graphics on them:

```
import org.opencv.core.MatOfRect;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.objdetect.CascadeClassifier;
import org.opencv.objdetect.Objdetect;
```

2. Create a new object named `faceDetector`, of the `CascadeClassifier` class:

```
CascadeClassifier faceDetector = new
CascadeClassifier();
```

3. You can load the correct cascade classifier with the `load` method, by providing the relative path to the `cascadeclassifier` file. We just use the filename, `haarcascade_frontalface_alt.xml`, because the file resides in the `SecurityCamera` directory. If you'd like to place the cascade classifier files in a separate directory, for example a directory named resources, use the path `resources/haarcascade_frontalface_alt.xml`:

```
faceDetector.load("haarcascade_frontalface_alt.xml");
```

4. The `detectMultiscale` method detects faces in every captured frame:

```
faceDetector.detectMultiScale(image,
faceDetections);

System.out.println(String.format("Detected
%s
faces",
faceDetections.toArray().length));
```

5. To draw a rectangle on the detected face, you can use the following code snippet:

```
for (Rect rect : faceDetections.toArray()) {
Imgproc.rectangle(image, new Point(rect.x, rect.y),
       new Point(rect.x + rect.width, rect.y + rect.height),
new Scalar(0, 255, 0), 2);
}
```

Build and run

1. First, rebuild the `SecurityCamera` project with NetBeans and run it from the Raspberry Pi using the following command:

```
sudo java -jar

/home/pi/RASPI3JAVA/SecurityCamera/dist/SecurityCamera.jar
```

2. An application window will open; click on the **Start** button to start the video capture. If the captured frame has a human face, the program will identify it and draw a rectangle around the face to highlight it. The process continuously detects and highlights faces in every captured video frame.

Figure 7-19 shows a single face captured by the application:

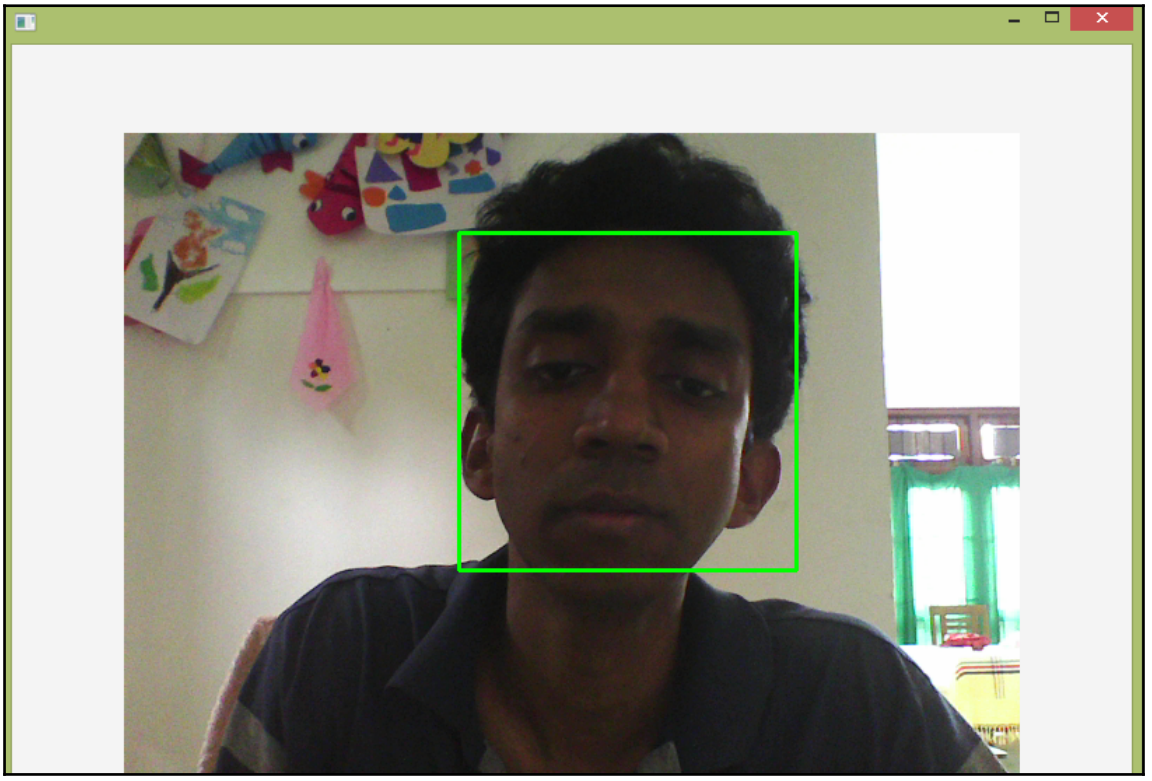


Figure 7-19: Single face capture

Figure 7-20 shows multiple faces captured by the application:

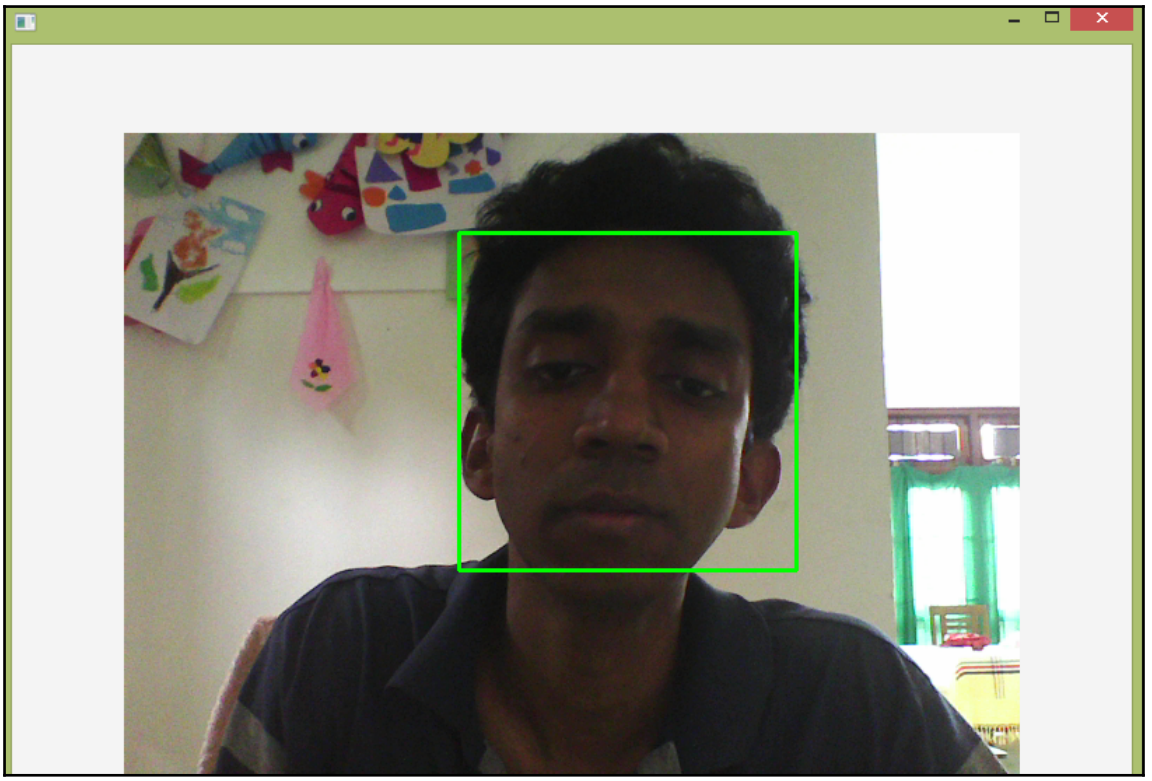


Figure 7-20: Multiple faces captured

3. Click the **Stop** button to stop the video capture.

Summary

In this chapter, you learned how to use the camera module with a Raspberry Pi to capture video as a series of frames, and process them to detect human faces using a cascade classifier, which is `haarcascade_frontalface_alt.xml`. You can improve this application by displaying the detected person's name, along with the highlighting rectangle, sending an email or SMS alert, or implementing a web stream to access the video through the network and display the video on a web page.

Index

G

H

I

J

K

L

M

N

O

P

R

S

T

V

W

Z