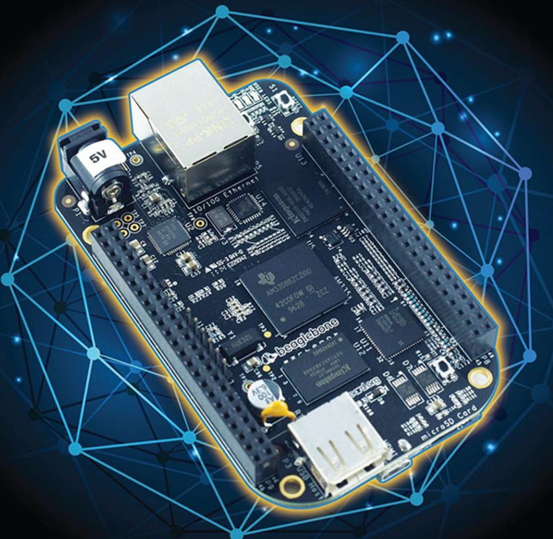
The BeagleBone Black Primer



QUE®

Brian McLaughlin

The BeagleBone Black Primer

Brian McLaughlin

Que

800 East 96th Street,
Indianapolis, Indiana 46240 USA

The BeagleBone Black Primer

ISBN-13: 978-0-7897-5386-1
ISBN-10: 0-7897-5386-3

Library of Congress Control Number: 2015946119

Printed in the United States of America

First Printing: September 2015

Editor-in-Chief
Greg Wiegand

Executive Editor
Rick Kughen

Development Editor
Ginny Bess Munroe

Managing Editor
Sandra Schroeder

Project Editor
Seth Kerney

Copy Editor
Bart Reed

Indexer
Tim Wright

Proofreader
Laura Hernandez

Technical Editor
Anton Olsen

Publishing Coordinator
Kristen Watterson

Interior Designer
Mark Shirar

Cover Designer
Mark Shirar

Compositor
Jake McFarland

Photography
Helene McLaughlin

Contents at a Glance

Table of Contents

About the Author

Brian McLaughlin is an engineer by profession and by hobby. Brian earned a bachelor's degree in computer science from North Carolina State University and a masters of engineering in systems engineering from the University of Maryland. With a solid foundation in software, Brian was initially exposed to more advanced topics in hardware while working on the Hubble Space Telescope Project. Over time, Brian began writing for GeekDad and has become a part of the growing Maker community. Brian lives in Maryland with his beautiful wife and two boys.

Dedication

For Mom & Dad

Acknowledgments

I wish I could acknowledge everyone who ever taught me something about STEAM (science, technology, engineering, art, and mathematics) topics, but that would be almost every teacher, instructor, mentor, and co-worker I have ever had to this point in my life. I would like to start by thanking the Integration and Test and software development teams for the Wide Field Camera 3 instrument on the Hubble Space Telescope, the first place I worked where the rubber met the road between hardware and software. I would like to thank my mentors—specifically Larry Barrett and Curtis Fatig—with whom I worked on the James Webb Space Telescope project and other projects. From them I was always learning something about engineering, working in a high-pressure environment, travelling the world, and finding out about life in general. I would like to thank my friends at GeekDad who helped me find a passion for writing about technical topics for fun and not just for my 9-to-5 job.

I would like to thank the people and companies who provided hardware and parts in support of this book including Tektronix, Oscium, SparkFun, and Element14.

I would like to offer my apologies to my neighbors in my little cul-de-sac in Columbia, Maryland. Writing a book while still holding down a full-time job was much harder than I had anticipated, and my lawn and yard suffered as a consequence. I promise I will keep them looking better!

I of course need to thank my parents, Glen and Diane, and my brother, Glen. Our parents always encouraged us to explore, learn, and grow, and my brother, in addition to sharing systems with me, showed me the Mosaic web browser before most people knew what the Web even was. I also need to thank my Uncle Lou, who passed along computers as he upgraded, and always made sure we were working on learning the basics of flying with *Flight Simulator*. It was also thanks to my parents and my Uncle Lou that I went to Space Camp in seventh grade.

Finally, I must acknowledge all of the rest of my loving family, particularly my beautiful wife, Helene, and my boys, Sean and Liam. Without everyone's support, patience, understanding, love, and patience, I never would have finished this book. (Yes, patience needed to be there twice.)

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email:	feedback@quepublishing.com
Mail:	Que Publishing ATTN: Reader Feedback 800 East 96th Street Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at quepublishing.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

The world is becoming a place where the traditional technical disciplines of science, technology, engineering, and mathematics (STEM) have boiled over into the world of art to produce STEAM (that is, STEM with Art thrown in the mix). It is the beginning of a new Renaissance. Just like in Da Vinci's time, cross-domain studies in all the STEAM topics are critical, and they are often unified via some form of electronics.

For example, an art installation may allow for a mechanical, interactive sculpture. This sculpture may require some "senses" to understand changes in the environment—everything from a change in temperature to the traditional senses of touch, sight, sound, taste, and smell. These changes are processed by some electronic means, and then some action is taken with that information. Maybe the head of a statue "looks" at you when you walk past.

For some, even the most technical work can seem like art. The layout of a circuit on a board, an elegant programming solution, and the RS-25 rocket engine, among many other technical solutions, are all like looking at art to me.

This book strives to provide the information necessary for you to find your own art in the world of STEAM. We will use a very accessible and powerful electronics board for this task—the BeagleBone Black.

Who This Book Is For

Targeting an audience for a book such as this can be tricky. For example, there are people in the artistic world—I've known many myself—who want to integrate electronics into their art projects, but they find the task daunting. I wrote this book so that those individuals can start to understand how electronics work and forge a path forward to bring their artistic creations to life.

There are others out there who have plenty of experience with electronics and building projects, but want to move toward using the power of the BeagleBone Black. For those readers, many of the sections of this book will provide quick reference to information on accessing the pins and functionality of the board and how to accomplish the basic tasks they need to know to build larger projects.

None of the projects are completely finished out in the course of the book. They are left at the level of the breadboard and still lack the finalized look of a project with a completed enclosure and installation. This is on purpose. I only give you enough information to be dangerous—and to go out on your own and build something amazing.

How This Book Is Organized

I try not to make any assumptions concerning what you may know about electronics and computers, other than basic familiarity with traditional desktop environments. With that in mind, I attempt to start off slowly. Here's what you'll find in the chapters of this book:

- **Chapters 1–5:** These chapters provide you with an overview of embedded electronics and development platforms. In these chapters, you learn what the BeagleBone Black represents and what major parts it is made from. You also learn how to procure a board, hook it up for the first time, and get something running. You learn some basics of electronics and how to get electrons to obey your will and desires. Well, they won't obey, but they will follow the paths you force them down and at the rates you desire. Finally, you learn how to use programming to make things happen on the board, and you get some exposure to a couple of the programming languages available and learn some of the differences between them.
- **Chapters 6–8:** These chapters provide some more advanced topics on hardware interactions and the environments you can use for operating your board. A number of operating system environments are available to run on the board, and in these chapters you learn how to switch out an operating system. You also learn some more low-level hardware interface information and about the ecosystem of standardized hardware expansion boards for the BeagleBone Black (known as Capes).
- **Chapters 9–14:** These chapters offer some insight into building more complex projects with the BeagleBone Black. You learn about how sensors work, build an environment-monitoring station for your working area, and find out how to manipulate items in your environment via motors. Finally, you get into various projects. You'll learn how to give your creator's vision, which can be used to actually track a person's face. You'll also tap into the computer in your car, and even listen to the data sent from aircraft so that you can track the aircraft in your area.
- **Chapter 15:** This chapter leaves you with some room to think about where you can go after reading the book. You'll learn about places to start expanding on what you have learned, a little about securing your BeagleBone Black, and some of my favorite project resources.

Only you know how much knowledge and experience you have as you begin reading this book. If you're completely new to this world, you will probably want to work through the book sequentially and build your knowledge as you go. If you're an experienced user who just wants examples of how to accomplish specific tasks with the BeagleBone Black, you will probably tend to bounce around and grab the bits you need.

Conventions Used in This Book

A couple of conventions are used throughout this book. Monospaced font calls out source code and terminal interactions. For example:

```
print "This is a line of source code"
```

```
~/bbb-primer/$ this is a terminal interaction
```

Note that source code is called out as a Listing, but terminal interactions are not. Also note that, in terminal interaction, content the user should enter is in **`bold mono font`**.

Let Me Know What You Think

If you want to contact me, feel free to email me at **bjmclaughlin@gmail.com**. I welcome questions that clarify points made in the book and constructive criticism.

However, as with many technical topics, there is often more than one way to accomplish a goal. Therefore, if you offer a comment that just shows a different way to accomplish the same thing or a quicker, more efficient way when the point of the example was obviously to be clear and thoughtful and not efficient, I will likely mentally acknowledge your point, archive the email, and move on with my life. Remember, my goal is to be as clear and accessible to as many levels of interest as possible.

With all of that said, I think you will enjoy the book, and I hope that you will learn how to accomplish whatever it is you set out to do after reading it. *Allons-y!*

This page intentionally left blank

3

Getting Started

To get started with the BeagleBone Black, you will want to obtain the board itself and some of the basic parts. The board is available from a number of resources. The beagleboard.org website provides some great places to buy the board (see http://beagleboard.org/black). Of course, I have my favorite suppliers, which are both a major part of the Maker community:

- **SparkFun Electronics (http://sparkfun.com)**—Boulder, Colorado–based SparkFun Electronics was founded on the idea of open source and open hardware. The company has an extensive catalog of electronics parts and components and consistently supports the community. SparkFun also has excellent tutorials and active forums where you can always find help on your projects. If you live in the Boulder area, be sure to watch for the in-person classes and other events the company sponsors!
- **Adafruit (http://adafruit.com)**—Based in New York City, Adafruit was founded by Limor "Ladyada" Fried, a true celebrity in the Maker community. The company is another great source for all the components and parts you might need for your projects and for taking the steps to move your projects beyond the introductory ones in this book. Adafruit also has an extensive tutorial section on its website.

When I am shopping for parts, I am often torn between these two suppliers. They are both excellent companies that have provided great support to me and countless others. Determining which to use, however, shouldn't be as much of a struggle as I might make it seem. The two companies work well together, participate in events together, and are both friendly. They represent the best of the community because they aren't there to compete against each other; they support the community and make sure we have what we need to get our projects off the ground. If you have questions on electronics basics or more complicated techniques, you can generally find a tutorial on one of the two sites or find willing support in the forums and via email.

In each chapter, I will outline the parts you will need for that chapter's project. We will take a practice run with this chapter and look at getting your BeagleBone Black set up and running. First, note some of the parts you'll need:

- BeagleBone Black
- USB cable (USB A to USB Mini B)
- +5 volt DC power supply (at least 1,000 milliamps)
- Ethernet cable

The USB cable should come with the BeagleBone Black; if it does not, make sure you inform your supplier. If a friend gave you the board, then just ask nicely. To get started, you also need a way to communicate with the board. In this chapter I'm going to discuss how to connect the BeagleBone Black to another computer and also how to connect remotely via Ethernet. Another option is to connect a monitor, keyboard, and mouse directly.

Setting Up and Saying "Hello, World!"

The big moment has arrived, and it is time to power up your BeagleBone Black and start working! We start with a direct computer connection via USB. This is a simple step. Simply plug the USB cable into the BeagleBone Black and the other end into a USB port on the computer.

As soon as the board is connected to your computer, you should see the lights on the board come to life. Four lights should start blinking on the board. These are four "user" lights, labeled USR0, USR1, USR2, and USR3, as shown in Figure 3.1. There is also a power light labeled PWR. The power light should stay constantly lit. The user lights will blink with different activities. At the default boot on a clean board, you will find the user lights configured as follows:

- **USR0**—This light blinks in a heartbeat pattern: two quick flashes, a pause, and then repeat.
- **USR1**—This light is configured to blink on activity from the microSD card. Because the board isn't plugged in a microSD card yet, you shouldn't see any activity on this light yet.
- **USR2**—This light flashes on CPU activity.
- **USR3**—This light flashes when the built-in flash memory is accessed. The default operating system should be installed in this embedded Multi-Media Card, or eMMC, memory, so you should see activity as the board is accessing the built-in, default operating system and file system.

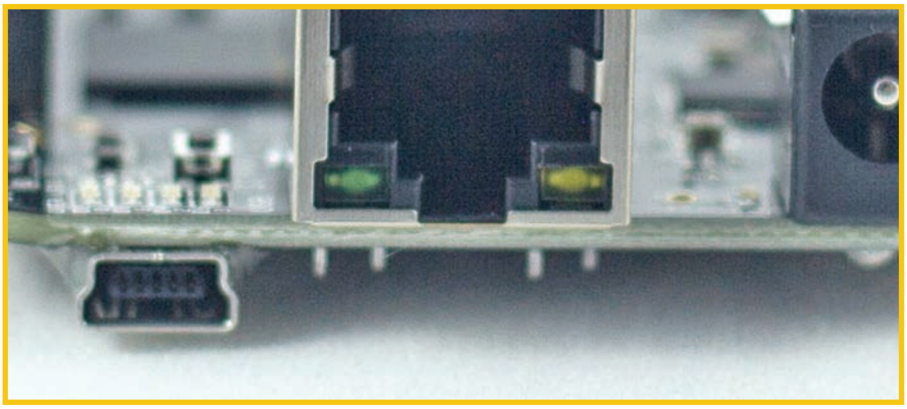


FIGURE 3.1 The BeagleBone Black user lights, power light, and USB port highlighted.

Next, you're going to have to install the drivers necessary for your computer to talk to the BeagleBone Black. On a Windows 7 laptop, you just plug the board in via USB to allow a drive to mount from the board that contains driver files. Use these files directly so that you don't have to look for them. You also have the option of downloading the drivers from the BeagleBoard organization website (http://beagleboard.org/getting-started). Drivers are available for Windows 32- and 64-bit environments, OS X, and Linux. You should refer to the specific setup instructions for your machine.

Now that you have your BeagleBone Black up and operating and have the drivers installed, what can you do? Your BeagleBone Black is already running a web server! You can use the Chrome or Firefox web browser to navigate to the board's web server at http://192.168.7.2.

NOTE

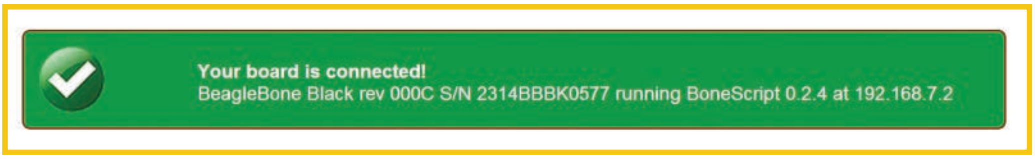
Web Browser Warning

The web server is not compatible with Microsoft Internet Explorer. Just stick with Chrome- or Firefox-compatible browsers. You really shouldn't use Internet Explorer anyway. For anything. Take this as a public service message. You can get Chrome or Firefox at the following links:

http://www.google.com/chrome/browser

http://www.mozilla.org/en-US/firefox/new

Navigate to the BeagleBone Black website at http://192.168.7.2. Your browser should present a very colorful and active website, and you should see something like the banner shown in Figure 3.2.



Your board is connected!
BeagleBone Black rev 000C S/N 2314BBBK0577 running BoneScript 0.2.4 at 192.168.7.2

FIGURE 3.2 Banner from the BeagleBone built-in website letting you know everything is working just fine.

This banner gives you information about your BeagleBone Black. First, the banner is green and has a check mark. That must mean everything is good, right? Also, the banner tells you that the board is connected. These are all good indications that all is right with your BeagleBone Black's world. Table 3.1 explains the other information that is supplied.

TABLE 3.1 Default Website Banner Information

Listed Information	Description
BeagleBone Black	You bought a BeagleBone Black, right? This is a good sign that you bought what you thought you were buying.
Rev 000C	This tells you that, in this case, version 000 of the revision C BeagleBone Black hardware is running.
S/N 2314BBBK0577	This is the serial number of the specific BeagleBone Black.
Running BoneScript 0.2.4	The board runs BoneScript version 0.2.4. BoneScript is a version of JavaScript for the BeagleBone environments.
At 192.168.7.2	This is the IP address of the board on the virtualized network over the USB connection. It should match the address you typed into the address bar of the browser.

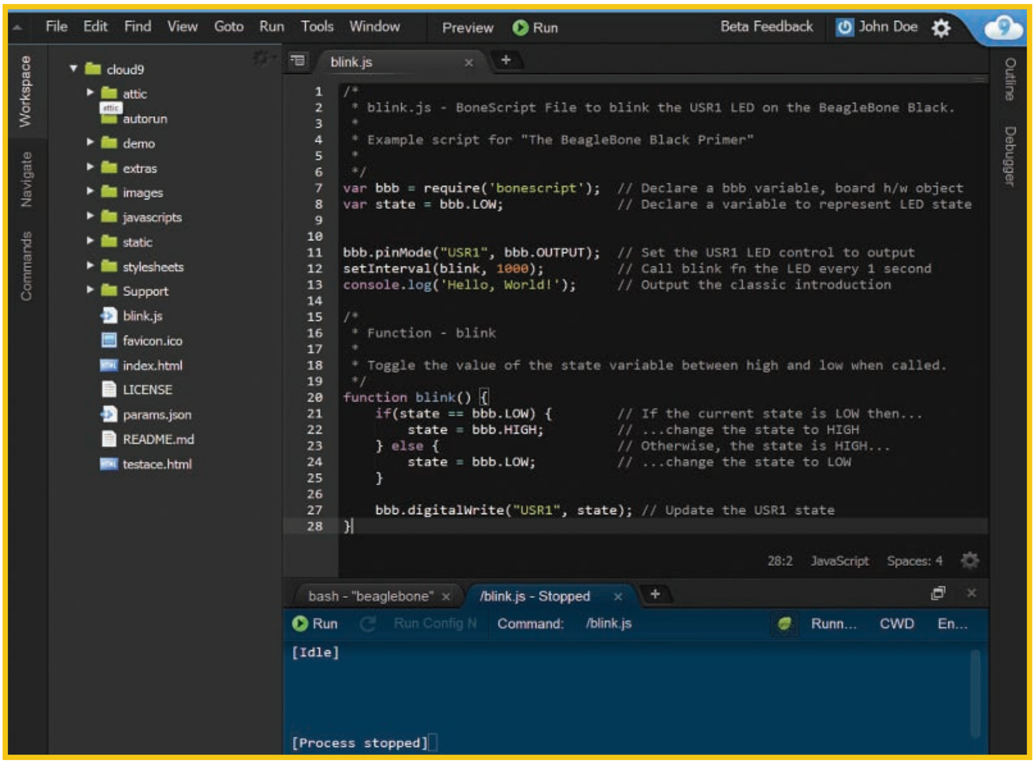
Congratulations! You've now successfully powered up, connected to, and communicated with your board! That was easy, wasn't it? Let's make a couple changes and use one of the user lights we discussed before to make it flash. About halfway down the page is a section titled "Cloud9 IDE" (IDE stands for *integrated development environment*). Click the header, and the Cloud9 IDE will launch in a new browser window or tab (see Figure 3.3). This is a powerful IDE running directly on the BeagleBone Black through a web interface.

So, what is an IDE? In short, an IDE is used as an all-in-one place where you can write software directly on the BeagleBone Black. It includes an editor, a way to execute code, and many other useful features.

When a person is just starting out with a new programming language, there is a tradition that the first program they write simply displays "Hello, World!" in some manner that fits into the environment. In many languages, this is accomplished by simply printing the message, whereas in some Windows environments, an alert message is displayed. That tradition has been extended into the hardware world with a program that makes a light blink once a second.

In our case, we are approaching a board for the first time and trying out a language for the first time, so why don't we try both displaying a message and blinking a light? Follow these steps to create a new file, write the code to accomplish our task, execute the code, and get blinking:

1. In the main window of the Cloud9 environment you'll find a + button. Click this button and select New File. This will open a blank text file where you can enter code. If there are other tabs open, you can close them. Feel free to peruse any "getting started" information on those pages.
2. Enter the code shown in Listing 3.1 into the document.
3. Save the file on the board. In this case, the file's name is blink.js.
4. In the environment, click the Run button.



```
/*
 * blink.js - BoneScript File to blink the USR1 LED on the BeagleBone Black.
 *
 * Example script for "The BeagleBone Black Primer"
 *
 */
var bbb = require('bonescript');  // Declare a bbb variable, board h/w object
var state = bbb.LOW;              // Declare a variable to represent LED state

bbb.pinMode("USR1", bbb.OUTPUT);  // Set the USR1 LED control to output
setInterval(blink, 1000);         // Call blink fn the LED every 1 second
console.log('Hello, World!');     // Output the classic introduction

/*
 * Function - blink
 *
 * Toggle the value of the state variable between high and low when called.
 */
function blink() {
    if(state == bbb.LOW) {        // If the current state is LOW then...
        state = bbb.HIGH;         // ...change the state to HIGH
    } else {                      // Otherwise, the state is HIGH...
        state = bbb.LOW;          // ...change the state to LOW
    }

    bbb.digitalWrite("USR1", state); // Update the USR1 state
}
```

```
[Idle]


[Process stopped]
```

FIGURE 3.3 The Cloud9 IDE running on the BeagleBone Black.

LISTING 3.1 blink.js

```
/*
 * blink.js - BoneScript File to blink the USR1 LED on the BeagleBone Black.
 *
 * Example script for "The BeagleBone Black Primer"
 *
 */
var bbb = require('bonescript');  // Declare a bbb variable, board h/w object
var state = bbb.LOW;               // Declare a variable to represent LED state


bbb.pinMode('USR1', bbb.OUTPUT);  // Set the USR1 LED control to output
setInterval(blink, 1000);          // Call blink fn the LED every 1 second
console.log('Hello, World!');     // Output the classic introduction

/*
```

```
16:  * Function - blink
17:  *
18:  * Toggle the value of the state variable between high and low when called.
19:  */
20: function blink() {
21:     if(state == bbb.LOW) {         // If the current state is LOW then...
22:         state = bbb.HIGH;          // ...change the state to HIGH
23:     } else {                        // Otherwise, the state is HIGH...
24:         state = bbb.LOW;           // ...change the state to LOW
25:     }
26:
27:     bbb.digitalWrite('USR1', state); // Update the USR1 state
28: }
```

It will take a couple of seconds, but the code will start executing. You should see a light just next to the heartbeat light blinking on for a full second and then off for a second. Success!

Let's step through the code you just blindly put into the environment and executed. Glad you trust me! The source starts with these six lines of code:

```
1:  /*
2:   * blink.js - BoneScript File to blink the USR1 LED on the BeagleBone Black.
3:   *
4:   * Example script for "The BeagleBone Black Primer"
5:   *
6:   */
```

These lines look fairly readable to a human and not like source code. That's because this code is what's called a *comment*. A comment starts with `/*` and ends with the `*/` and includes everything in between. The extra asterisks at the beginning of the other lines are just to make things look good. There is another way to signify comments in BoneScript/JavaScript, and that is using `//`. These are used to describe what is occurring on a line of code. Everything from the `//` to the end of the line is a comment. Comments are not executed or even seen for execution. You will see a couple of different styles of commenting in different languages throughout the book.

Line 7 accesses a shared library of source code, called `bonescript`, that is provided to you as part of the environment:

```
7:  var bbb = require('bonescript');  // Declare a bbb variable, board h/w object
```

This code accomplishes many tasks behind the scene that you don't need to worry about for now. Access to the library is assigned to variable `bbb`. This means that we can use the variable `bbb` to access resources in that special library, as you will see on the following lines:

```
8:  var state = bbb.LOW;             // Declare a variable to represent LED state
```

Line 8 declares another variable called `state`. We are going to use `state` to track whether we set our signal for the USR1 light to `HIGH` or `LOW`. When the state is set to `HIGH`, the voltage on the electronics attached to that light is set to +5V, and it is set to 0V for `LOW`. When the voltage is set `HIGH` at +5V, the electrical potential on the light is increased, which means the light can do work. What happens when a light can do work? It lights up!

Something important to remember here is that setting the `state` variable to `HIGH` or `LOW` doesn't actually change the power supplied. We do that using a function called `digitalWrite`, which is a part of the `bonescript` library we can now access through the `bbb` variable. More on that function later. Now we hit a line that does something with the electricity on the board:

```
11: bbb.pinMode('USR1', bbb.OUTPUT);  // Set the USR1 LED control to output
```

With this line, we are calling a function called `pinMode`, which is part of the `bonescript` library, and using another `bonescript` library constant called `OUTPUT`. This means we are configuring the USR1 pin to output the voltage rather than sensing a voltage from somewhere else in a circuit. In total, this line says, "Take the pin attached to the light USR1 and get it ready to output, please."

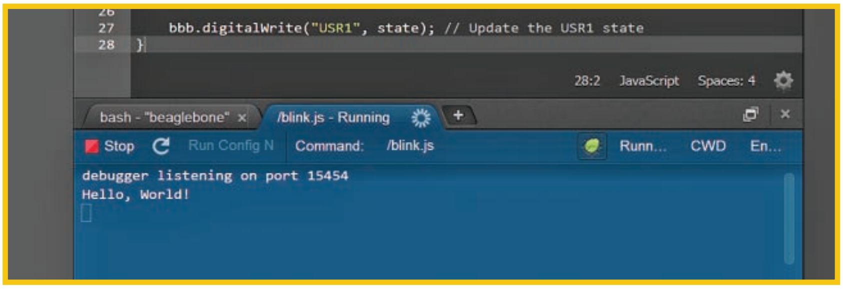
The next line utilizes a function called `setInterval` to run the meat of the program:

```
12: setInterval(blink, 1000);          // Call blink fn the LED every 1 second
```

This line of code tells the system to execute the function `blink` once every second. Line 13 has nothing to do with blinking our light. This is a simple statement that prints our classic first-time program announcement out to a console:

```
13: console.log('Hello, World!');      // Output the classic introduction
```

In the Cloud9 IDE environment, you will see this printed on a lower tab labeled "/blink.js – Running," as shown in Figure 3.4.



```
    bbb.digitalWrite("USR1", state); // Update the USR1 state
}
```

```
debugger listening on port 15454
Hello, World!
```

FIGURE 3.4 The "Hello, World!" statement written to the console log.

The final lines define a function called `blink`. This function simply checks the status of the `state` variable and changes it to the opposite state. This function is called once every second by the `setInterval` function. The real meat of the function is on line 27. The call to `digitalWrite` makes the actual change to the hardware to change the status of the physical circuit attached to the USR1 light:

```
27:      bbb.digitalWrite('USR1', state); // Update the USR1 state
```

That is all the code required to use BoneScript to blink a light and print a message to the console! It is important to remember that BoneScript is defined only by the `bonescript` library. The underlying syntax and structure is just JavaScript, a scripting language used in many places on the Web. This means that you can use JavaScript tutorial and reference resources to help you understand or to get any clarification.

For simple examples throughout the book, I will stick to BoneScript just to make it easy. For more complex code and functionality, I use other programming languages such as C/C++ and Python. I will comment the code to help with readability if you are not familiar with those languages; however, I highly encourage you to seek out other resources to learn those languages in depth because that is not the focus of this book. The next chapter will delve into some more complex development with BoneScript and the Cloud9 IDE to enable your own explorations. It will also introduce you to the basics of programming with other languages.

Connecting to Ethernet

Thus far, we have talked to the BeagleBone Black through a USB connection to a computer. This is all fine and well, but the power of the BeagleBone Black is that it's a standalone computer capable of working on its own. Our next step is to cut the cord from our computer and connect the BeagleBone Black to a network.

In accomplishing this, we can drop the USB cable from our setup and exchange it for an Ethernet cable. We also need to power our board. Ethernet, unlike USB, does not provide power in normal configurations. There's an option called Power over Ethernet, abbreviated PoE, but this is not a normal network configuration, so we will assume you need a separate power supply. I purchased the power supply I am using from SparkFun. It has a 5V output and can provide up to 1A of current.

Most home networks use a system called Dynamic Host Configuration Protocol (DHCP). In this configuration, a component on a network is assigned an address on the network automatically. To make this easy on ourselves, we know when the BeagleBone Black is connected via USB that it has an address of 192.168.7.2. We can use this to our advantage and connect to both Ethernet and USB at the same time and see what address our board is assigned for the Ethernet connection. So, with your board already connected to the USB, plug your Ethernet cable into the board and into your network.

When you connect the Ethernet cable, you should see the lights on the Ethernet port of your board light up. This means you've connected! Now, to see the IP address that has been assigned, we are going to break out to a new piece of software and connect via Secure Shell (SSH). We are about to delve into the world of the Linux command line.

There are many ways to connect via SSH. If your computer runs Linux or OS X, getting to a terminal is as easy as opening a Terminal session. The commands to use are the same as you would see working on the command line of a Linux or OS X machine. Following is an example of connecting via SSH from the Linux command line first. The process is the same for OS X. We will get to Windows in a moment.

From the Terminal, execute the following command:

```
[brian@mercury-fedora-vm ]$ ssh root@192.168.7.2
```

With this command, you ask the system to use the `ssh` command to connect as `root` to the computer at `192.168.7.2`, which we know is our BeagleBone Black's USB connection. When you execute this command and you have all the connections set right, you are presented with the following prompt:

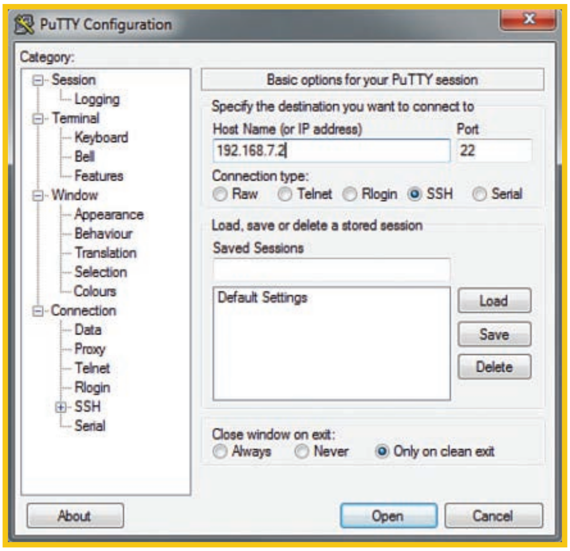
```
The authenticity of host '192.168.7.2 (192.168.7.2)' can't be established.
ECDSA key fingerprint is c0:81:1a:f4:58:b9:51:15:00:df:ee:71:c4:d9:fd:54.
Are you sure you want to continue connecting (yes/no)?
```

This prompt is associated with security (referring to the first *S* in SSH). Your computer has never connected to this host via SSH before, and it wants to validate that this is the computer you mean to connect with. What does this buy you? It allows you to be sure that the computer you are connecting to in the future is the one you intended to, with no hacker interference. You should accept this by entering **`yes`**. The `ssh` program will let you know that it has accepted the security key and that it is added to the list of known servers.

```
The authenticity of host '192.168.7.2 (192.168.7.2)' can't be established.
ECDSA key fingerprint is c0:81:1a:f4:58:b9:51:15:00:df:ee:71:c4:d9:fd:54.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.7.2' (ECDSA) to the list of known hosts.
```

Now, you request to log in as the root user. The root account is a powerful account and should be password protected by default, but the BeagleBone Black has a blank root password by default. We will change this later before we set the board up to be on a network doing a job.

In the Windows environment, I recommend PuTTY for SSH connections. It is easy to find with a Google search, and installation is a breeze. When you start the application, you are presented with the configuration window shown in Figure 3.5. Notice in the hostname that I've entered the USB assigned address of your BeagleBone Black, 192.168.7.2. Just below the Host Name text entry, you select the connection type, which is SSH in this case. Once you've entered these settings, click the Open button.



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.168.7.2
Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

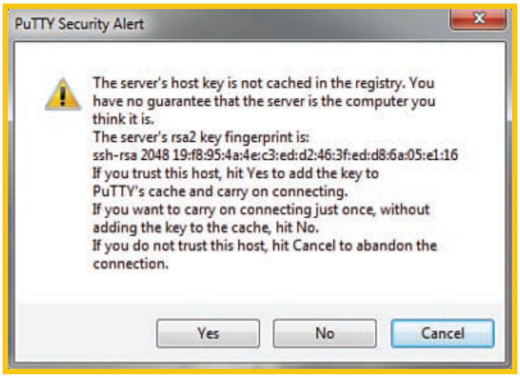
Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

FIGURE 3.5 The PuTTY Configuration window.

Another window will pop up that looks a lot like the information you saw the first time you tried to connect in the Linux terminal, and it serves the same function (see Figure 3.6). Click the Yes button to accept the security key and continue by logging on.



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 19:f8:95:4a:4e:c3:ed:d2:46:3f:ed:d8:6a:05:e1:16
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

FIGURE 3.6 The PuTTY Security Alert window.

From here on, regardless of the operating system or Terminal application you are using, the output will be the same. That is because what you'll see now is actually on the BeagleBone Black.

From now on, if you log in with SSH via the USB default connection, you will not see the prompt for the security key. The session will now present you with the following command prompt:

```
root@beaglebone:#
```

This is the default prompt for the default user. If you are familiar with Linux or a similar operating system, then you'll know you're in a Bash shell. The information provided by the prompt can be very useful and even customized. The information in Table 3.2 is presented in the default prompt.

TABLE 3.2 Default Prompt Information

Prompt Information	Description
`root`	The information in this first block tells us the username for the shell. In our case, we logged in as root.
`beaglebone`	The hostname, on the network, we are logged in to.
	This represents the directory we're currently working in on the file tree. In this case, the tilde is shorthand for the user's home directory.

Now, we are going to enter our second command. This command, called `ifconfig`, is used to report the current network status of the system. Let's go ahead and enter it and then see the response:

```
root@beaglebone:# ifconfig
eth0      Link encap:Ethernet  HWaddr 7c:66:9d:58:bd:41
          inet addr:192.168.1.161  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::7e66:9dff:fe58:bd41/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4059 errors:0 dropped:2 overruns:0 frame:0
          TX packets:147 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:616100 (601.6 KiB)  TX bytes:18322 (17.8 KiB)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usb0      Link encap:Ethernet  HWaddr e6:8c:89:9a:b6:c8
          inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
          inet6 addr: fe80::e48c:89ff:fe9a:b6c8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1717 errors:0 dropped:0 overruns:0 frame:0
          TX packets:136 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:200409 (195.7 KiB)  TX bytes:31059 (30.3 KiB)
```

The results provided tell us about three different network adapters represented on the system: `eth0`, `lo`, and `usb0`. We can ignore `lo` for now; it's the local loopback connection. The two of interest to us are `eth0` and `usb0`. The default USB connection is `usb0`. There are a lot of fields here, but the field we are interested in is labeled `inet addr`. Here is the `usb0` interface information again, with that field highlighted in bold:

```
usb0      Link encap:Ethernet  HWaddr e6:8c:89:9a:b6:c8
          inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
          inet6 addr: fe80::e48c:89ff:fe9a:b6c8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1717 errors:0 dropped:0 overruns:0 frame:0
          TX packets:136 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:200409 (195.7 KiB)  TX bytes:31059 (30.3 KiB)
```

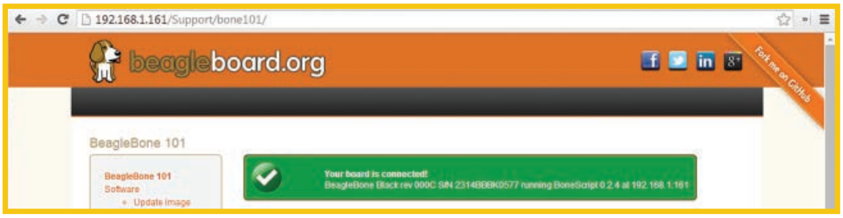
The value associated with this address should look familiar. It is the same address we used to access the website and to log in to the board. Now, what we are looking for is the address that has been given to the board via DHCP. The Ethernet port is represented by interface `eth0`, and by looking at its `inet addr` field, we know that, in this case, the DHCP has assigned the board an address of `192.168.1.161`, as shown here:

```
eth0      Link encap:Ethernet  HWaddr 7c:66:9d:58:bd:41
          inet addr:192.168.1.161  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::7e66:9dff:fe58:bd41/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4059 errors:0 dropped:2 overruns:0 frame:0
          TX packets:147 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:616100 (601.6 KiB)  TX bytes:18322 (17.8 KiB)
          Interrupt:40
```

Unless you have changed the IP address space in your network, and if you have I trust that you know what you are doing with your network, your board will have an IP Address in one of the public IP Address spaces, either 192.168.x.x or 10.0.x.x.

Now, with the board connected to Ethernet and assigned an address on the network, you can unplug the USB connection and plug in the +5V power adapter. You've now put your BeagleBone Black on the network, independent of the computer you were using before. You are ready to be an active member of your home's network ecosystem!

Let's check the website connection via your network connection. Using your browser, navigate to the address provided earlier for `eth0`. In the case of my network, that's 192.168.1.161, as you can see in Figure 3.7.



Your board is connected!
BeagleBone Black rev 000C S/N 2314BBBK0577 running BoneScript 0.2.4 at 192.168.1.161

FIGURE 3.7 Banner from the BeagleBone built-in website letting you know everything is working just fine—this time, via the Ethernet connection.

Now, let's log in via SSH to the `eth0` connection. It is going to look familiar:

```
[brian@mercury-fedora-vm ]$ ssh root@192.168.1.161
The authenticity of host '192.168.1.161 (192.168.1.161)' can't be established.
ECDSA key fingerprint is c0:81:1a:f4:58:b9:51:15:00:df:ee:71:c4:d9:fd:54.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.161' (ECDSA) to the list of known hosts.
Debian GNU/Linux 7

BeagleBoard.org BeagleBone Debian Image 2014-04-23

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
Last login: Fri Jul 18 15:06:44 2014 from mercury-win.local
root@beaglebone:#
```

As you can see, it is the same process for logging in as before, but with a different address. This example is from a Linux machine, but the process is identical as the one we used previously for PuTTY, but with a different address.

Something interesting to note is that if you run the `ifconfig` command, the `usb0` adapter is still available. That's because we haven't actually changed anything in the operating system configuration; we just used a different connection. The USB option is still there waiting for us to connect, and it will be unless we turn it off in the operating system. We will get into more detail about the underlying operating system and some of the options we have for alternate operating systems on the BeagleBone Black. In our next chapter, we delve into the hardware and discuss some basics of electronics that will be necessary for many of the remaining chapters.

This page intentionally left blank

Index