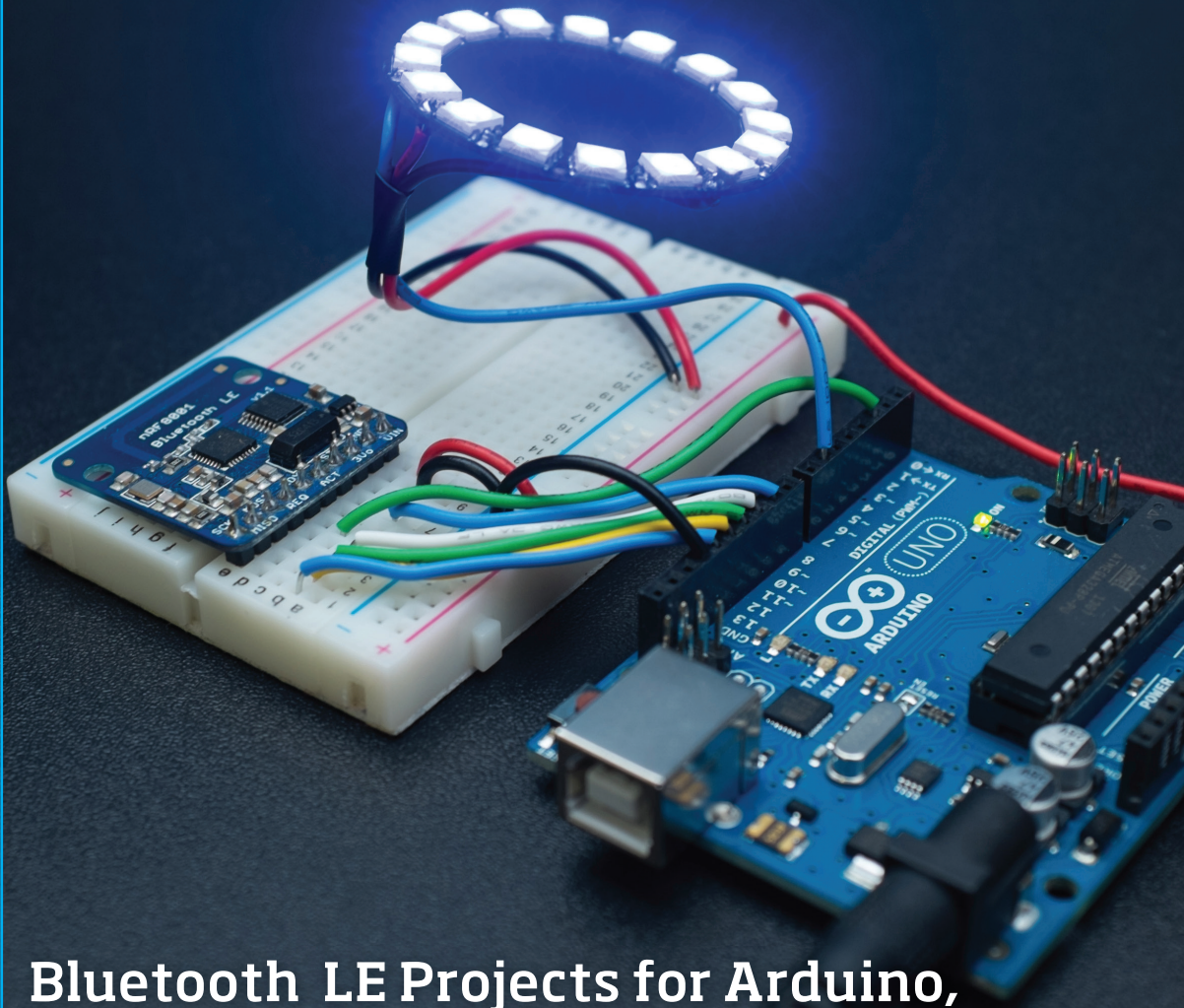


# Make: Bluetooth



Bluetooth LE Projects for Arduino,  
Raspberry Pi, and Smartphones

Alasdair Allan, Don Coleman &  
Sandeep Mistry

# Make: Bluetooth

*Bluetooth LE Projects for Arduino, Raspberry Pi,  
and Smartphones*

Alasdair Allan, Don Coleman,  
Sandeep Mistry



## **Make: Bluetooth**

by Alasdair Allan, Don Coleman, and Sandeep Mistry

Copyright © 2016 Alasdair Allan, Don Coleman, and Sandeep Mistry. All rights reserved.

Printed in the United States of America.

Published by Maker Media, Inc., 1160 Battery Street East, Suite 125, San Francisco, CA 94111.

Maker Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact O'Reilly Media's institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editor:** Roger Stewart

**Production Editor:** Melanie Yarbrough

**Copyeditor:** Gillian McGarvey

**Proofreader:** Christina Edwards

**Indexer:** Last Look Editorial

**Interior Designer:** David Futato

**Cover Designer:** Jean Tashima

**Illustrator:** Rebecca Demarest

December 2015: First Edition

### **Revision History for the First Edition**

2015-11-24: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781457187094> for release details.

Make:, Maker Shed, and Maker Faire are registered trademarks of Maker Media, Inc. The Maker Media logo is a trademark of Maker Media, Inc. Make: Bluetooth and related trade dress are trademarks of Maker Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Maker Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

978-1-457-18709-4

[LSI]

---

# Table of Contents

---

<b>Preface</b> .....	<b>ix</b>
<b>1. Introduction</b> .....	<b>1</b>
Talking About Bluetooth LE .....	1
Protocols and Profiles .....	3
The GAP .....	3
The GATT .....	3
Services and Characteristics .....	3
UUIDs .....	4
An Example Service .....	5
Making Sure Your Machine Has Bluetooth LE .....	6
OS X .....	6
Apple iOS .....	7
Linux .....	7
Android .....	7
Microsoft Windows .....	7
What Haven't We Told You About Bluetooth LE? .....	7
<b>2. Getting Started</b> .....	<b>9</b>
The Arduino .....	9
The Board .....	9
Powering the Board .....	11
Input and Output .....	11
Communicating with the Board .....	11
Installing the Arduino IDE .....	11
Installing on OS X .....	12
Installing on Linux .....	13

Installing on MS Windows .....	13
Connecting to the Board .....	13
Installing the BLE Peripheral Library .....	15
Setting Up Raspberry Pi .....	16
Installing BlueZ .....	17
Verifying the Bluetooth LE .....	17
Node.js .....	17
Installing Node.js .....	18
On Linux and Raspberry Pi .....	18
Installing Libraries with npm .....	18
Setting Up Dependencies for noble and bleeno .....	18
OS X .....	18
Raspberry Pi and Linux .....	18
Installing PhoneGap .....	19
PhoneGap Developer App .....	19
Bluetooth Low Energy Plugin .....	19
PhoneGap Versus Cordova .....	20
Platform Tools .....	20
Android .....	20
<b>3. Smart Light Switch .....</b>	<b>23</b>
What Is a Smart Switch? .....	23
Hardware .....	23
The Breadboard .....	24
Getting Started .....	25
Resistor Color-Coding .....	25
Blinking an LED .....	26
Adding a Switch .....	31
Software Debouncing .....	35
Making a Real Light Switch .....	36
By Changing the Hardware .....	36
Changing the Software .....	37
Adding Bluetooth .....	39
Wiring Up the Adafruit Bluefruit LE Module .....	39
Modifying Our Sketch .....	41
Testing the Service .....	46
Using Real Lightbulbs .....	47
Conclusion .....	51
<b>4. BLE Lock .....</b>	<b>53</b>
Lock Service .....	53
Hardware .....	54
Lock Software .....	56

Programming .....	56
Setup .....	57
Loop .....	58
Unlock Characteristic Written .....	59
Open Lock .....	59
Reset Lock .....	60
Testing the Lock .....	62
iOS .....	62
Android .....	64
Mobile Application .....	65
Callbacks .....	66
BLE Lock App .....	66
CSS .....	67
HTML .....	67
JavaScript .....	69
Run the App .....	78
Improving the Lock .....	79
<b>5. Bleno Lock .....</b>	<b>81</b>
Hardware .....	81
Lock Software .....	86
Libraries .....	86
Programming .....	86
Conclusion .....	92
<b>6. Weather Station .....</b>	<b>93</b>
Hardware .....	94
Libraries .....	96
Programming .....	96
Compile and Upload .....	101
Serial Monitor .....	101
Using the Service .....	101
Using LightBlue on iOS .....	102
Using nRF Master Control Panel on Android .....	102
PhoneGap .....	104
Create the Project .....	104
HTML .....	105
JavaScript .....	107
Run the App .....	115
What's Next? .....	116
<b>7. NeoPixel Lamp .....</b>	<b>117</b>
Hardware .....	117

NeoPixels .....	117
Building the Hardware .....	117
Software .....	120
LED Service .....	120
Programming the Arduino .....	121
Generic Bluetooth Client .....	128
Building a Phone App .....	129
Create the Project .....	129
HTML .....	129
CSS .....	131
JavaScript .....	131
Run the App .....	137
Enhancements .....	138
Physical Switch and Dimmer .....	138
Lamp .....	148
<b>8. SensorTag Remote .....</b>	<b>153</b>
Hardware .....	153
Create the Project .....	155
SensorTag and Noble .....	155
SensorTag Remote .....	161
A Simpler Version .....	163
Next Steps .....	164
Arduino Simple Key Service .....	165
<b>9. HID over GATT .....</b>	<b>169</b>
HOGP and BLEPeripheral .....	169
Volume Knob .....	170
Hardware .....	171
Arduino Library Setup .....	173
Testing the Rotary Encoder .....	173
Implementing the Volume Knob .....	174
Conclusion .....	180
<b>10. Beacons .....</b>	<b>181</b>
What You'll Need .....	181
iBeacon .....	182
What Data Does an iBeacon Advertise? .....	182
Building and Detecting a Beacon .....	183
Creating a Mobile App that Uses iBeacons .....	189
Eddystone Beacons and the Physical Web .....	197
What Data Does an Eddystone Beacon Advertise? .....	198
Building and Detecting Your Own Beacon .....	200

Conclusion .....	203
<b>11. Drones .....</b>	<b>205</b>
What You'll Need .....	206
Testing Out the Drone .....	207
Controlling the Rolling Spider with Node.js .....	207
Setting Up the Project .....	207
Discovering the Drone .....	207
Getting Started: Basic Takeoff and Landing .....	208
Keyboard Control .....	210
Conclusion .....	217
<b>12. Going Further .....</b>	<b>219</b>
The Arduino .....	219
Hardware Suggestions .....	219
Further Reading .....	222
<b>Appendix A. HID Over GATT Pairing .....</b>	<b>223</b>
<b>Index .....</b>	<b>233</b>



---

# Preface

---

One of the drivers behind the recent explosive growth in the Internet of Things has been the Bluetooth Low Energy (LE) standard. What makes it so appealing is the ubiquity of smartphones—both Apple and otherwise—with support for the standard, which means that thing-makers no longer have to worry about a display or user interface. And that means that things like smart lightbulbs can look a lot more like lightbulbs rather than a computer that happens to have a light attached to it.

Bluetooth LE is very different from classic Bluetooth—in fact, pretty much the only thing that is the same is the name. You’re probably used to thinking about radios as sort of like serial connections working similarly to a phone call between two phones—once you establish a connection, each person talks as the other listens and vice versa. They stay connected, even if neither is saying anything, until one hangs up and the call is ended.

In systems like these, data is transferred using a queue, and when data is read by the receiver, it’s erased from the queue—just as once my words reach your ears over the phone, they’re out of the communications channel. Effectively, this is how classic Bluetooth works.

## What Makes Bluetooth LE Unique?

---

Instead of communicating via a point-to-point connection like a phone, a *Bluetooth LE radio* acts like a community bulletin board, with each radio acting as either a board or a reader of the board.

If your radio is a bulletin board—called a *peripheral device* in Bluetooth LE parlance—it posts data on its board for everyone in the community to read. If your radio is a reader—called a *central device* in Bluetooth LE terms—it can read from any of the boards (the peripheral devices) that have information it cares about.

If you don't like that analogy, you can also think of peripheral devices as the servers in a client-server transaction. Similarly, central devices are the clients of the Bluetooth LE world because they read information from the peripherals.

## But I Like Serial Connections!

Most (perhaps all?) of the Bluetooth LE radios breakout boards available to makers right now—the RedBearLab BLE mini and the Adafruit Bluefruit LE, for instance—pretend to look like serial devices for simplicity's sake and present a UART service to the user. Effectively these radios are “faking” old-style serial communication on top of the underlying bulletin board paradigm. It's a hack, and not a bad one.

*Serial over Bluetooth LE* makes the transition from classic Bluetooth to Bluetooth LE easier for people who are used to Serial Port Profile (SPP) and UART. However, there's a downside. It doesn't take advantage of Bluetooth LE. These Bluetooth serial services are just stuffing data across generic transmit and receive pipes. If a device uses serial, it also needs to define a protocol for the data that is sent and received.

Bluetooth LE offers device makers the ability to create Bluetooth LE devices with self-describing services. If the characteristics are well designed and the descriptors make sense, you can use services without documentation. (An example of this is using a Smartbotics Lightbulb with the LightBlue iOS app.)

Contrast this with other devices using SPP, like services where you need to learn the details of which bytes to send over the wire to turn on an LED and change its color. These devices need really good documentation and/or additional libraries to do simple stuff like set the color of a LED.

Returning to our bulletin board example, we're creating a board (the *service*) that has a sticky note attached (known as a *characteristic* in Bluetooth LE parlance), which we can read, letting us know if the LED is on or off, or write to—allowing us to control the LED.

## Building a Custom Service

---

Unfortunately, until recently, building custom services for Bluetooth LE has been fairly complicated and not for the faint-hearted. However, it's getting simpler as several good tools now exist to do most of the heavy lifting for you.

In light of that, we decided to look at one platform—the Nordic Semiconductor radios—and figure out a complete toolchain that would allow you to build a custom service for those radios, and make use of that service from an Arduino project. We picked this particular radio because it's readily available and there is good library support.

This book is the result.

## Who Should Read This Book?

---

This book provides an introduction to the topic of how to build and deploy Bluetooth LE sensors and devices. It takes a hands-on approach and teaches you how to build things and how to use them, not just the protocols and architecture behind the standard.

If you're a programmer or a maker who wants to get started with Bluetooth LE, this book is for you.

## What You Should Already Know

---

This book is intended as an introduction to working with Bluetooth LE, and it assumes a technical background. However, we walk you through installing and using all the software and development environments you'll need, including how to get started with the Arduino, Raspberry Pi, Node.js, and PhoneGap.

## What You Will Learn

---

This book will guide you through building a series of connected projects—from lightbulbs, to locks, to beacons, and drones. It will walk you through your first hardware prototypes, show you how to improve them, and teach you how to build Bluetooth LE connected devices for the Internet of Things.

## What's In This Book

---

### *Chapter 1, Introduction*

This chapter talks about the Bluetooth LE standard and the concepts you'll need to know about before you can start building Bluetooth LE devices.

### *Chapter 2, Getting Started*

This chapter walks you through setting up the tools and software you'll need to set up your development environment.

### *Chapter 3, Smart Light Switch*

This chapter shows you how to build a smart light switch that not only lets you turn the light on or off using the switch but also remotely via Bluetooth LE. The switch knows its current status—in other words, whether the bulb is on or off—and will send out a notification if that status changes.

### *Chapter 4, BLE Lock*

This chapter shows you how to build a lock that can be opened using your phone via Bluetooth LE. We will also walk through writing a mobile app that will run on iOS or Android using PhoneGap, which will control the lock.

### *Chapter 5, Bleno Lock*

This chapter re-creates the Bluetooth LE lock built in the previous chapter but this time using Node.js on the Raspberry Pi. It uses the same lock service as the original Bluetooth LE lock so it can be controlled using the same mobile app.

### *Chapter 6, Weather Station*

This chapter walks you through building a Bluetooth-LE-enabled weather station that can measure temperature, humidity, and pressure.

### *Chapter 7, NeoPixel Lamp*

This chapter will show you how to build an RGB lamp that is controllable from your phone using an Arduino and a NeoPixel Ring with 16 LEDs.

### *Chapter 8, SensorTag Remote*

This chapter shows you how to turn a TI SensorTag into a remote control for your computer, triggering an action on your computer when one of the SensorTag buttons is pressed.

### *Chapter 9, HID over GATT*

This chapter explores the Human Interface Device (HID) profile and shows you how to build a Bluetooth LE volume control.

### *Chapter 10, Beacons*

In this chapter we explore Bluetooth LE beacons, showing how to create them using Node.js and detect them using smartphones.

### *Chapter 11, Drones*

This chapter shows you how to control a Parrot Rolling Spider drone over Bluetooth LE using a computer and Node.js.

### *Chapter 12, Going Further*

This chapter provides a collection of pointers to more advanced material on the topics we covered in the book, and material covering some of those topics that we didn't manage to talk about in this book.

## **Conventions Used in This Book**

---

The following typographical conventions are used in this book:

### *Italic*

Indicates new terms, URLs, email addresses, filenames, and file extensions.

### Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

### Constant width bold

Shows commands or other text that should be typed literally by the user.

---



*This element signifies a general note, tip, or suggestion.*

---



*This element indicates a warning or caution.*

---

## Using Code Examples

---

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from Make: books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission. All the code from this book is available on GitHub at <https://github.com/MakeBluetooth>.


Support material for the book is also available at [the book's website](#).

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Make: Bluetooth* by Alasdair Allan, Don Coleman, Sandeep Mistry (O'Reilly). Copyright 2016 Alasdair Allan, Don Coleman, Sandeep Mistry, 978-1-457-18709-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at [bookpermissions@makermedia.com](mailto:bookpermissions@makermedia.com).

## Safari® Books Online

---

 **Safari**® *Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.*

---

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of [plans and pricing](#) for [enterprise](#), [government](#), [education](#), and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds [more](#). For more information about Safari Books Online, please visit us [online](#).

## How to Contact Us

---

Please address comments and questions concerning this book to the publisher:

Make:  
1160 Battery Street East, Suite 125  
San Francisco, CA 94111  
877-306-6253 (in the United States or Canada)  
707-639-1355 (international or local)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://bit.ly/make-BT>.

Make: unites, inspires, informs, and entertains a growing community of resourceful people who undertake amazing projects in their backyards, basements, and garages. Make: celebrates your right to tweak, hack, and bend any technology to your will. The Make: audience continues to be a growing culture and community that believes in bettering ourselves, our environment, our educational system—our entire world. This is much more than an audience, it's a worldwide movement that Make is leading we call it the Maker Movement.

For more information about Make:, visit us online:

- [Make: magazine](#)
- [Maker Faire](#)
- [Makezine.com](#)
- [Maker Shed](#)

To comment or ask technical questions about this book, send email to [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com).

## Acknowledgements by Alasdair Allan

---

Everyone has one book in them, but this isn't mine. Depending how you count them, this is my ninth book. But every book is different, and they do not write themselves. So I'd like to thank my co-authors Don Coleman and Sandeep Mistry, and my editors at Make:, Brian Jepson and Roger Stewart, for holding my hand throughout the process.

I very much want to thank my wife, Gemma Hobson, for her continued support and encouragement. Those small, and sometimes larger, sacrifices an author's spouse routinely has to make don't get any less inconvenient the second, or third, or the n'th time around. I'm not sure why she lets me write—perhaps because I claim to enjoy it so much. Thank you, Gemma. Finally to my son Alex who, seven years on from my first book, is actually almost old enough to read this one.

## Acknowledgements by Don Coleman

---

I'd like to thank my wife, Meghan, and son, Liam, for their support and patience during the many hours I spent working on this book. Tom Igoe was very helpful by working through ideas and testing early versions of code and libraries with his students. Guan Yang's work on Arduino nRF8001 provided a huge kickstart in getting custom BLE services running on Arduino without proprietary tools. Brian Jepson, Roger Stewart, and the team at Make: deserve a shout-out for their patience with us and their persistence in getting this book published. I appreciate the support of my colleagues at Chariot Solutions, LLC. Lastly, it's been awesome working with my talented co-authors Alasdair and Sandeep.

## Acknowledgements by Sandeep Mistry

---

This is the first book I've been involved in. I would like to thank my co-authors Alasdair Allan and Don Coleman, and the editors Brian Jepson and Roger Stewart for guiding me through the process and providing valuable feedback throughout its production.

# Introduction

# 1

In [Chapter 2](#), we'll install the tools we need to write code and deploy services. But before we do that, we need to talk about Bluetooth LE, and make sure your machine can talk to it, too.

This chapter talks about some of the jargon you'll need to understand to work with Bluetooth LE. The following chapter walks you through installing all of the software and configuring the hardware you'll need for the projects in this book. After working through these two chapters, you should probably proceed with [Chapter 3](#), which is a solid introduction to working with Bluetooth and the Arduino. However, after that you should be able to pick and choose among the rest of the projects in the book.

## Talking About Bluetooth LE

---

Bluetooth LE divides the world into peripheral and central devices. *Peripheral devices* are things like sensors; they're typically small, low-powered, and resource-constrained. *Central devices* are things like mobile phones and laptops, although these can usually also operate in peripheral mode.

Peripherals can operate in two modes: either by broadcasting or being directly connected to a central device. The broadcast mechanism is one of the big differences between Bluetooth LE and classic Bluetooth; it allows data to be sent out by the peripheral to any device in range.





*The real-world range of a Bluetooth LE device depends on the transmitting power of the radio. Since higher transmitting power means more battery is required, Bluetooth LE is, unsurprisingly, a short-range standard. While it's perfectly possible to have a real-world range of greater than 30m (about 100ft), a more typical operating range is between 2 and 5m (around 5 to 15ft).*

---

This means that a Bluetooth LE peripheral device doesn't necessarily need to be "paired" with a central device in order to transfer data. In Bluetooth LE, we speak of it as "connected" rather than paired as we did with Bluetooth 2.1. In broadcast mode, the peripheral will periodically send out advertising packets, available to any device that's looking for them, for devices acting as "observers."

The standard *advertising packet* describes the broadcasting device and its capabilities but is also capable of including custom information—sensor data, for instance—that you want to broadcast.

Broadcasting data from your peripheral is a good choice if you're building something like a weather station where the data isn't sensitive. However, there is no provision for security when broadcasting, so for personal data, the central device should connect to the peripheral, not the other way around.

Connections are exclusive. This means that a peripheral cannot be connected to more than one central device at a time. When a central device connects to a peripheral, the peripheral will stop advertising itself, and other devices will not be able to see it or connect to it until the first connection is terminated. However, whereas a peripheral can only be connected to one central device, a central device can be connected to more than peripheral at the same time.

---



*The Bluetooth 4.1 specification removed the restriction that peripheral devices can only be connected to a single central device. Going forward, a peripheral can be connected to multiple central devices simultaneously. However, there are still many devices and chipsets that remain limited in this fashion.*

---

If you need to exchange data between the peripheral and the central device, then you need to establish a connection between the two devices.

## Protocols and Profiles

---

On top of the protocols that make up the Bluetooth LE standard, the specification defines what are called *profiles*. These are either the basic modes of operation needed by all Bluetooth LE devices, like the Generic Access Profile and Generic Attribute Profile, or *profiles covering specific use cases* such as the Heart Rate Profile.

### The GAP

The *Generic Access Profile* (GAP) defines roles for devices, including the peripheral and central roles we mentioned in the last section, alongside advertising and discovery.

There are two ways to advertise data using GAP: advertising data and scan response packets. While both packets use the same payload format, and consist of up to 31 bytes of data, only the advertising data packet is mandatory. It is sent out at a preset advertising interval—the longer the interval, the less power used—on receipt listening devices can request the scan response packet with additional data if it exists.

Using custom advertisement data in the broadcast packets is how both the iBeacon and Eddystone standards are implemented; see [Chapter 10](#) for more information.

Once a connection with the peripheral has been made, you will use GATT services and characteristics to communicate with the peripheral device, and advertising will stop until the connection is terminated.

### The GATT

The *Generic Attribute Profile* (GATT) defines how Bluetooth LE transfers data back and forth between peripheral and central devices. It defines profiles, which are collections of services. Each service has characteristics, which contain data.

Roles change when moving from GAP to GATT. GATT defines two roles: client and server.

It may seem backwards, but peripheral devices are known as *GATT servers* and the more powerful central devices are *GATT clients*. Think of it this way: the server has data and the client wants data. All connections between the devices are initiated by the client.

After connecting, the client can get a list of services offered by the server. Before connecting, the central device had a potentially incomplete list of services from the advertising data.

### Services and Characteristics

*Services* are used to break up the data into logically associated chunks, and consist of a collection of characteristics. *Characteristics* are the containers that hold the data associated with a service. Both services and characteristics are identified by a unique identifier, known as a UUID; see “UUIDs”.

Characteristics contain at least two attributes: a *characteristic declaration*, which contains metadata about the data, and the *characteristic value*, which contains the data itself. The

characteristic can also contain additional descriptors to expand on the meta data. Together, the declaration, value, and any optional descriptors form a bundle than make up a characteristic.

Characteristics can be defined as read or write. Characteristics are read by the client using a read request, with the returned value of the request being the characteristic value. Characteristic values can be written using a *write request*. The server returns a confirmation after the value is written. There is an additional write property called the *write command*. When a characteristic value is written with a write command, the server does not send any response back to the client. The write command is sometimes called write without response.

Two additional properties are *notify* and *indicate*. Both of these are server-initiated communication. A client subscribes to be notified when a characteristic's value changes. When a change occurs, the server notifies the client by sending the new value. An indication is similar to a notification, except that the client must acknowledge the receipt of the indication.

Characteristics can have multiple properties. For example, one characteristic could allow read, write, and notify.

## UUIDs

Bluetooth uses Universally Unique Identifiers (UUIDs) for many things, including services and characteristics. Bluetooth services that [have been approved by the Bluetooth Special Interest Group](#) are assigned 16-bit UUIDs. All other services and characteristics must use 128-bit UUIDs, which can be generated with tools such as `uuidgen`, as shown here:

```
$ uuidgen
437121E5-A6F0-43F9-8F8F-4AB73D6CC3EB
```

In this book, we use 16-bit UUIDs. Technically we're breaking the rules, but for us it's a lot easier to type DCF8 than 391CDCF8-4BD4-4507-BE23-B57DFD1F870B. See [Table 1-1](#) for a list of officially approved 16-bit UUID ranges.

**Table 1-1** *Table 1-1. Officially approved 16-bit UUID ranges*

UUID	Description
00xx	Namespace Descriptors
18xx	Services
27xx	Units
28xx	Declarations
29xx	Descriptors
2Axx	Characteristics

For the LED examples, we reuse SmartBotics' service for their [RoboSmart lightbulb](#). For the button examples, we reused Texas Instruments' [Simple Key Service](#) from their Sensor Tag. For the thermometer examples, we create some 16-bit UUIDs.

It's fine to reuse UUIDs if existing services and characteristics meet your needs. If you're making your own services, use 128-bit UUIDs. See [the Bluetooth Developer Site](#) for more information.

## An Example Service

Pulling this together, let's look at a typical example service for a lightbulb, as shown in [Table 1-2](#) and illustrated in [Figure 1-1](#).

This is part of the service definition advertised by a [RoboSmart Light Bulb](#). As you can see, despite not being an officially approved service definition, like many other manufacturers, RoboSmart uses 16-bit UUIDs.

**Table 1-2** *Lightbulb Service (FF10)*

Characteristic	UUID	Property	Value	Comments
Light switch	FF11	Read, Write	1	1 on, 0 off
Dimmer setting	FF12	Read, Write	0x7F	0x00 to 0xFF
Power consumption	FF16	Read	340	Watt Hours

This is part of the service definition advertised by a RoboSmart Light Bulb (see [Figure 1-1](#)).

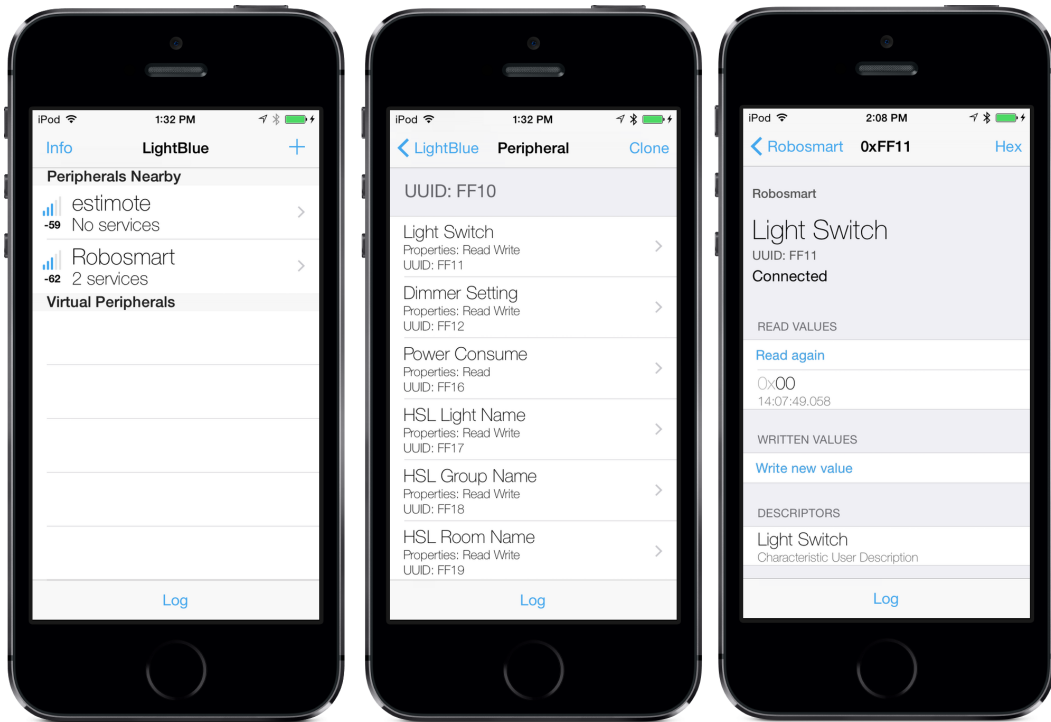


Figure 1-1 Exploring the Lightbulb service

## Making Sure Your Machine Has Bluetooth LE

Support for Bluetooth LE is available on most major OS platforms. However, if your laptop or desktop hardware doesn't support Bluetooth LE, there is a large number of Bluetooth LE USB dongles available. Many are based around the same Broadcom BCM20702A0 chipset; for this book, we used the [IOGEAR GBU521 Bluetooth 4.0 USB Micro Adaptor](#), which can be picked up for about \$10 to \$15 on Amazon and other online retailers. USB adapters with the CSR8510 chipset also work well.

### OS X

All Apple laptop and desktop machines built after 2011 have Bluetooth LE hardware built-in, though if you're using a Mac built before that and running Mountain Lion or later, you're probably still okay. You can pick up a Bluetooth LE USB adaptor and it should work out of the box.

If you're running a pre-Mountain-Lion distribution of OS X, it's still possible to get most Broadcom-based Bluetooth LE adaptors to work, but it's probably going to be a lot harder. For Lion, you'll have to go into the `/System/Library/Extensions/` folder and make changes inside the `IOBluetoothFamily.kext`, adding the current Product ID and Vendor ID

values of your USB dongle before reloading the kernel extension. For those running Pre-Lion distributions of OS X, things are even more difficult.

## Apple iOS

Support for Bluetooth LE on the iPhone and iPad has existed since iOS 5, although we'd recommend iOS 7 as the minimum version because it introduced iBeacon support to the operating system. Hardware support for Bluetooth LE was introduced with the iPhone 4s, iPad (third generation), and the iPod touch (fifth generation). All current Apple iOS devices support the standard.

## Linux

Linux uses the BlueZ service to talk to Bluetooth devices, and Bluetooth LE has been supported since version 4.93.

Linux users should take a look at the instructions in [“Setting Up Raspberry Pi”](#) where we talk about installing the BlueZ service to work with the Bluetooth 4.0 USB adaptors.

## Android

Support for Bluetooth LE in the core Android OS was introduced in Android 4.3, although version 4.4 introduced numerous bug fixes and is probably the minimum recommended version. Before this, though some manufacturers shipped hardware that was Bluetooth-LE-compatible, there was no software support, which caused many manufacturers to introduce their own libraries to support it—which were unfortunately all incompatible.

## Microsoft Windows

Microsoft Windows 8 and above has built-in support for Bluetooth LE. Unfortunately, Windows XP, Vista, and Windows 7 only support Bluetooth 2.1, and while it's theoretically possible to get these earlier versions to talk to Bluetooth LE devices—at least for Windows 7, it's an extremely challenging problem that's well beyond the scope of this book. If you're working with Bluetooth LE under Windows, you should at least be running Windows 8.

## What Haven't We Told You About Bluetooth LE?

---

Lots. The Bluetooth LE specification is a sprawling mess of [interlocking documents](#) that runs to thousands of pages; the [core standards document](#) is over 2,700 pages on its own. What we've given you here is the barest outline, a sketch, of how the standard works.

However, it's enough that you should now be able to go and confidently build projects using Bluetooth LE to actually do things in the real world. You can pick up the architecture and theory behind the protocols as you go along. But you have enough to get started.