O'REILLY®

Building Applications with iBeacon

PROXIMITY AND LOCATION SERVICES
WITH BLUETOOTH LOW ENERGY

Matthew S. Gast

O'REILLY®

Building Applications with iBeacon

High-precision location information is increasingly useful for mobile application developers, since it allows devices to interact with the world around them. This practical book shows you how to achieve *arm's reach* accuracy with iBeacons, simple transmitters that enable your applications to react to nearby surroundings and then deliver timely, relevant information—especially indoors, where GPS and cell service are inaccurate.

Whether you're enabling a map, giving users directions, creating a game, recommending purchases, letting users check in, or creating an immersive experience, you'll learn how iBeacons provide precise location information, empowering your applications to engage and interact with users nearby.

- Get examples of several application types you can build with iBeacons
- Learn how iBeacons provide applications with proximity information
- Set up, activate, and test iBeacons on both specialized and general-purpose hardware
- Explore the APIs and tools you need to develop location-aware mobile applications
- Use built-in iOS features to interact with iBeacons, including Passbook
- Build networks to help shoppers, travelers, conference attendees, and others find what they're looking for

Matthew S. Gast is the director of product management at Aerohive Networks, responsible for the software that powers its networking devices as well as investigating emerging technologies. He's an industry expert who led the development of IEEE 802.11-2012 and security task groups at the Wi-Fi Alliance, and the author of several books on Wi-Fi for O'Reilly, most recently *802.11ac: A Survival Guide*.

ELECTRONICS/NETWORKING

US $24.99 **CAN $26.99**

ISBN: 978-1-491-90457-2

Twitter: @oreillymedia
facebook.com/oreilly

Building Applications with iBeacon

Matthew S. Gast

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Building Applications with iBeacon
by Matthew S. Gast

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly 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 our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Brian Sawyer and Mike Loukides
Production Editor: Matthew Hacker
Copyeditor: Kiel Van Horn
Proofreader: Charles Roumeliotis
Indexer: WordCo Indexing Services, Inc.
Interior Designer: David Futato
Cover Designer: Ellie Volckhausen
Illustrator: Rebecca Demarest

October 2014: First Edition

Revision History for the First Edition
2014-09-23: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491904572* for release details.

978-1-491-90457-2

[LSI]

For A.,

A steady keel, cutting deep through seething waters,
A reminder to trim the sails, not howl at the unruly wind,
And a lighthouse, guiding me always home.

Table of Contents

Preface vii

1. Introduction 1
Why iBeacons? 2
Applications 5
Bluetooth Low Energy Beacons 9

2. The iBeacon Protocol 13
Bluetooth Basics of an iBeacon 13
iBeacon Advertising Packet Contents 16

3. Setting Up Your Own Beacons 19
Types of Hardware 19
Activating a Beacon 22

4. Application Development 31
Limitations on iBeacons 32
Basic iBeacon Programming Functions 33
Advanced iBeacon Programming Functions 37
Security and Privacy 38

5. iOS and iBeacons 41
iBeacon Development on iOS with Core Location 41
iBeacon Monitoring with Core Location 43
Ranging 48

6. Building iBeacon Networks. 53
Planning and Project Objectives 53
Beacon Location Selection 56
Project Checklist 59

Index. 61

Preface

We all carry supercomputers in our pockets. Sometimes we even still make phone calls with them, too. The computing power available in most smartphones today is vastly greater than what many desktop computers had when I started working with computers, and that additional computing power has been put to good use in redefining the way that we interact with the world.

Extending the mobile computing experience beyond the glass touchscreen is a key component of enabling the Internet of Things, developing wearable computers, and embedding intelligence in the world. Humans have sensors (such as eyes!) to interact with the world around us, and phones now have the electronic equivalent by using proximity technology to discover the immediate world around them.

But how can a phone recognize what it is *near*?

The answer to that question is the essence of proximity and helps move *nearby* interactions onto the smartphone touchscreen. Getting a basic readout on what is in the neighborhood is the most basic operation for making a phone part of a mobile computing system.

Or, to say it better, knowing what you are near is foundational. Proximity is the "Hello, world" for the Internet of Things.

Many technologies exist to help phones interact with the world around them. This book is about iBeacons, a Bluetooth technology that helps a device understand its location and surroundings with a high degree of accuracy. iBeacons enable a device to display web pages, control nearby machines, and negotiate transactions, all based on being near enough for actions to matter.

Who This Book Is For

This book is intended mainly for application developers. As you'll see, the protocol is straightforward. The main talent required to successfully use beacons is imagination to see how to apply proximity to the problems your application faces. iBeacons are cheap enough to easily get started with application development, and the cost is low enough that any organization can afford the tiny up front cost. In many cases, it is common to start off with just a single developer investigating the technology.

How to Use This Book

This book is organized into the following chapters:

Chapter 1, Introduction
: This chapter introduces the concepts of proximity sensing, which is the core of what beacons enable, and describes sample applications of the technology.

Chapter 2, The iBeacon Protocol
: Beacons describe the space around them using a simple protocol. Instead of directly describing the space, a beacon is used as a pointer by an application running on a mobile device to map the physical world containing the beacon into something that the application can act on.

Chapter 3, Setting Up Your Own iBeacons
: The beacon protocol is simple enough that it can be implemented straightforwardly in software. Applications can run on a general-purpose operating system such as Mac OS X, free applications on mobile devices, USB dongles, and even special-purpose tiny computers, such as Raspberry Pi or Arduino.

Chapter 4, Application Development
: After you have an iBeacon running, an application will watch for it and react to it. To find iBeacons, applications can *monitor* for transmissions and, once found, can choose to perform *ranging* operations to determine the distance to an iBeacon. Once identified, applications can also interact with web services or local storage. During the application development process, you will also have to consider the security of the application system

from end to end, which might require providing security beyond what is built into the protocol.

Chapter 5, iOS and iBeacons
: This chapter describes what is needed to get an application working on iOS. iOS extends the Core Location framework to use iBeacons, which enables developers to activate pre-programmed actions in response to detecting a beacon.

Chapter 6, Building iBeacon Networks
: The final step in creating an application is to place beacons in the field to support the application, so that the application works as desired.

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.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

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 O'Reilly 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.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Building Applications with iBeacon* by Matthew S. Gast (O'Reilly). Copyright 2015 Matthew S. Gast, 978-1-4919-0457-2."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

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:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

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/building_apps_w_ibeacon*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Writing a book is a collborative effort between author and editor. In many cases (including this book), it is unlikely that I would have finished the book as quickly without the persistence and drive of my editor, in this case Brian Sawyer. Brian helped the project in many ways, and the comparison of location and proximity benefited especially from his assistance.

One of the many reasons that writing for O'Reilly is worthwhile is the ease with which we can call on other experts in the field to participate in the technical review. My review team consisted of Doug Thompson, proprietor of the BEEKn blog; Carles Cufí, an engineer with Nordic Semiconductor; and Robert Davidson of Ambient Sensors. Doug brought his wide-ranging expertise and keen understanding of the market to bear. Carles and Robert are both highly experienced engineers with a deep understanding of Bluetooth, and they helped the book more clearly ground iBeacon firmly within the Bluetooth ecosystem. Naturally, any errors that remain in the book are my own and in no way reflect on the review team.

I am indebted to several people in the industry whom I met while writing the book and who graciously offered their assistance. I first met David Helms, the chief product officer at Radius Networks, when I was in Washington, DC, for a business trip and attended an iBeacon meetup. Over the next several months, David was a wonderful resource who was always willing to talk about industry developments. Our conversations were consistently inspiring and informative, and they helped this book in more ways than I can count. Trung Nguyen and Chi-Lang Ngo, both at Passforce, showed me great hospitality when I visited them in London and had a series of great conversations about how iBeacons worked within the Passbook ecosystem.

CHAPTER 1
Introduction

Close don't count in baseball. Close only counts in horseshoes and hand grenades.
—Frank Robinson

Close doesn't always count when building mobile applications. Mobile devices typically have smaller screens and might have limited forms of text input. But the user profile of a device is not the major constraint on its function. At its best, an application that knows where you are will augment reality to help you navigate and interact with the physical world.

We have seen some tentative steps toward applications that interact with the world around them. Star maps use location information and accelerometer-derived tilt readings to highlight stars in the sky on a real-time picture from a phone's camera.[1] Inexpensive GPS receivers have turned every smartphone into an up-to-date navigation system for drivers, turning archaic practices such as knowing how to fold a paper map or memorizing how to get around a city[2] into stories that we will someday scare children with.

1 Reading the tilt of a phone is a complex process that involves multiple sensors. Accelerometers measure the orientation of the phone relative to gravity, and gyroscopes provide a rate of change in the angles of the device around its axis. Multiple sensors can be combined in software (a process called *sensor fusion*), blending inputs from GPS, magnetic fields, acceleration, gyroscopes, and even barometric pressure.

2 The most famous test of navigation skill and geographic awareness is "The Knowledge," required of every London cab driver.

Until now, no technology has enabled you to interact with the world within "arm's reach." That has changed with the introduction of Bluetooth *proximity beacons*, often referred to as *iBeacons*.[3] Beacons bridge the gap between the data world of ones, zeros, and the glass screen on a mobile device and the physical world that users move through with their devices. By transmitting an identifier, a beacon can define a small area that devices react to, and those reactions can be used to create new applications and new interactions.

An iBeacon is a Bluetooth Low Energy proximity beacon that is submitted for compatibility testing under an Apple licensing program, and the Bluetooth proximity functions on Apple systems are referred to in developer documentation as iBeacon frameworks.

Much of the enthusiasm for these new applications comes from building systems for retailers, but the simple beacon protocol can be applied to many types of spaces, leading to varied applications as diverse as indoor direction finding, queue management, museum guides, and restaurant management.

Why iBeacons?

There are two interrelated reasons for the excitement around iBeacons. Most importantly, the technology enables a device to make extremely precise determinations of what is nearby. Even under ideal conditions, GPS technologies struggle to do better than a few meters, and GPS is often limited indoors. iBeacons can enable a determination within centimeters.

Equally as important, the high-precision location information is readily available to applications, which enables developers to create new types of experiences by incorporating this information into applications.

3 Particularly astute readers may be asking why we need proximity beacons if a typical phone has multiple sensors. The answer, as pointed out by one of my reviewers, is that the typical gyroscopes used in mobile devices are too noisy to perform high-quality navigation functions; hence, beacons are required to overcome the limitations of existing sensor technology.

iBeacons offer high precision *micro-location*, along with the ability to act on what a mobile device is near. No other technology yet offers that combination.

Location Versus Proximity

iBeacons define "what you are close to," which is often related to "where you are," but not necessarily. Many people use the terms *location* and *proximity* interchangeably. Though the words are related, they have subtle differences that make them distinct concepts.

Most importantly, *location* is an absolute, unchanging concept, typically defined by some form of a coordinate system. Whether you define it as geographical coordinates such as latitude and longitude, a street address, or even much larger distance scales such as a city, state, province, or country, a location is a fixed point or area of space. Many technologies can help you answer the question of "What is my location?" One of the most common ways to answer that question is to use GPS or cell tower mapping, and the resulting location often has the precision of a building or a street address. With this form of location, you can drive or walk to a building, find nearby events or restaurants, or check in for your visit.[4]

In contrast to the absolute of location, *proximity* is the answer to the question "What is nearby?" Rather than an absolute location, proximity is defined in relation to some other point.[5] Conceptually, proximity is a distance, and has no direction. Am I near another person? A particular display in a store? The conductor on the train?

Part of the confusion comes from the fact that proximity technologies can be used to provide location information. If a proximity network is installed in an indoor location where GPS does not work well, the reference points used by the proximity system may be used to help an application infer a location. That is, the proximity system

4 With all the technology development that has occurred in my career, it is sometimes hard for me to remember that when I first moved to Silicon Valley, I found an apartment by getting a daily list faxed to me and looking up street addresses on paper maps using the index of street names!

5 In more formal language, location has vector attributes, because it includes direction. Proximity is a scalar, because it has no direction.

knows that you are a given distance from three proximity points with known locations and, therefore, you are standing near baggage claim carousel 5.

When used in this manner, proximity technologies are said to be providing *micro-location*, because they can provide much more accurate information than GPS, especially in buildings where GPS does not operate with high accuracy (or possibly even at all). In a micro-location scenario, proximity can operate as a tighter description of where you are, but that misstates the power of proximity.

iBeacons are exciting because they can do more than provide location information. Proximity is based on "what is near you" instead of "where you are." As a result, an application can tell whether you are near an item of interest, regardless of its location.

For example, if you are visiting the Louvre, you might care whether you are near the Venus de Milo or Canova's Cupid's Kiss, regardless of the location of those artworks in the galleries. Using proximity technology means that an electronic museum guide knows what is near by knowing proximity information, rather than the more complicated process of the museum guide knowing where it is and where displays are and calculating proximity based on comparing the two points.

Proximity enables an application to identify something of interest based on what it is near, and that is often independent of location. As an example, consider a train passenger. The passenger's location is constantly changing as the train moves along the tracks, but an application that holds a ticket does not care about location. In contrast, it might be interested in knowing only when a conductor is nearby, so that it can display an electronic ticket.

Proximity services are based on a different set of technologies than location services. Rather than being large outdoor systems like cell networks or satellites, proximity is based on shorter-range technologies like Wi-Fi or Bluetooth Low Energy (the BLE-based iBeacon, for instance). Table 1-1 illustrates some of these key differences.

Table 1-1. Differences between location and proximity

Location	Proximity
GPS, Wi-Fi	iBeacon, Bluetooth, Wi-Fi Time Difference of Arrival (TDOA)
Driving directions	In-building directions
10s or 100s of meters[6]	Centimeters to meters

An important technical difference between the two classes of technology is that proximity technologies can be used indoors to provide the same types of services as large outdoor systems but on a smaller scale.

Interactivity

iBeacons define small regions in space, based on how far their signals move through space. The beacon transmissions can trigger a mobile device to take action at a precise location, typically through an application.

Rather than depend on a user to locate himself on a map, or press buttons, or select from a menu of actions, applications can now be triggered by the physical world.

Applications

The most important difference between proximity and location is that the smaller scale of proximity drives closer interaction between customers and a business. Although the technology is most discussed in retail, it is potentially useful in a wide variety of contexts.

Broadly speaking, applications use proximity technologies to improve location awareness in areas where technologies traditionally have not supported a fine-grained location. When iBeacons are used, they provide proximity information to enable the application,

6 When used outdoors, most mobile GPS devices have accuracy on the order of 10–15 meters, with substantially less accurate indoor resolution.

though, as we will see, the iBeacon provides a trigger to the application and works in conjunction with a companion web service.

Indoor Location and Proximity

The go-to technologies used for location, GPS and Wi-Fi, are not useful for indoor direction finding. That is, they provide location services, but not proximity. Possible applications for indoor proximity include:

Map replacement
: Within a large building, it can be difficult to find your way around without extensive maps. Large hospitals often are a challenge to navigate, as are malls. In both types of locations, a series of beacons can help a device accurately place itself on a map and augment reality by providing clear directions to a destination. Electronic maps have the additional advantage of being able to update electronically and instantaneously in response to changes in the physical location of departments, stores, or other points of interest.

Transit assistance
: Transit hubs such as airports and train stations can be confusing for infrequent travelers. (Navigating Shinjuku Station in Tokyo or Penn Station in New York City can be challenging for even highly experienced travelers.) Applications can already access ticketing information and use that to automatically assist travelers in finding a boarding gate, ending with calling up the relevant ticket at the end of the journey.

Indoor direction finding
: In heavily built-up cities, directions will often include a subway or train station exit to take for the optimal path to a destination. Both the Hong Kong and Tokyo subways use clearly identified exits with examples of major landmarks. By using proximity technology indoors, direction-finding apps could incorporate instructions to train riders to find the correct exit, even deep below ground.

Where is my car?
: This is a special case of indoor direction finding. In massive parking garages, it can be difficult to remember where you parked your car. GPS is often limited in parking structures given the robust construction needed to handle the weight of

vehicles. Instead of the mobile device giving you directions to a fixed point, it can guide you directly to your parking spot.

Museum guides
: Instead of printed paper guidebooks, museums can build apps for patrons. In addition to direction finding, major exhibits can have extended information available, allowing visitors to engage more deeply with major exhibitions than a few signs would allow. One of my favorite museum guides is at the Musée de la Musique in Paris, where an audio guidebook enables visitors to hear many of the musical instruments in the collection being played.

Retail store enhancement
: I've placed this item last on the list because it is the most obvious and likely to be familiar even without reading this book. Proximity allows a mobile device to take the display you are standing in front of into account, rather than just mere presence in the store. One of the earliest applications of beacons in retail is the official Apple Store app for iOS, which will welcome you to an Apple store and offer assistance with items you are standing in front of. By further developing this channel, it is possible to monitor shoppers through the store and offer promotions based on purchase history.

iBeacons are about much more than indoor locations. In addition to offering information about where a device is to help the user, the proximity of a device to a given location can also be used to trigger an action on the device.

Proximity-Triggered Actions

Location technologies can provide a position that is accurate to within a building, but some applications might benefit from even more accurate position information. Proximity enables actions to be taken when two devices are close to each other. Typically, this is discussed as a mobile device moving through space, but it might also be two mobile devices moving relative to each other.

Here are some examples of possible actions triggered by proximity:

Mobile advertisements
: Proximity is discussed so frequently with mobile ads that the two seem inextricably linked sometimes. When a device gets

close to a display, an advertisement can be pushed to the display. The display might be a coupon, notification of a sale, or simply an offer to receive more information.

Ticket validation
: On many trains, conductors are required to validate tickets, and one of my favorite developments in recent years is the use of easily integrated mobile ticketing services that allow me to carry tickets electronically and update them seamlessly. If the train conductor's equipment signals that the conductor is coming, the proximity of a passenger's mobile device to the train conductor can be used to display the ticket.

Treasure hunt
: At CES in 2014, an app rewarded users who went around the show floor to collect "virtual stamps." After getting near all nine virtual locations, users could then take the completed virtual card to the show office for a reward.[7]

Patient information integration
: Relentless checking and rechecking of treatment orders is a practical approach to reducing medical errors. Low-energy Bluetooth hardware is inexpensive enough that iBeacons on patient ID bands can be used to automatically pull up records on a doctor's tablet and confirm that patients are being treated appropriately.

Triggering actions based on proximity can bring devices into your business processes. One of the most common business processes to integrate users into is some sort of queue management, because many organizations have their customers use a line of some sort.

Queue Management

Many people now carry mobile devices with them as they move throughout their day, and the ability to interact with mobile devices enables many organizations to manage queues of customers more effectively in the following ways:

7 For a striking but less uplifting application of the same sort of proximity information, in April 2014, New York City's New Museum simulated a minefield using iBeacons.

Queue measurement
: By placing proximity beacons at the start and end of a queue, businesses can measure how long a typical transaction takes and can even expose that information to other customers. The California DMV, for example, regularly publishes queue times at different DMV offices (though they gather the information through methods other than beacons). Bank teller queues or airport security queues are often of great interest to customers.

Restaurant table pager
: Restaurants often use wireless devices to signal you that your table is ready. The existing devices are pagers and therefore have limited communication abilities. By using proximity technologies, an app running on a phone can "check in" to a queue for a table and potentially signal the restaurant to alert the patron when a table is ready. In this case, beacon technology is used to trigger the start of the interaction, but other data networks are required to complete it.

Transaction completion in retail
: Quick-service restaurants are managed for turnaround time. By using beacons, they can trigger an app to wake up and pay for a transaction automatically using a payment system such as PayPal or Square. Or if a restaurant allows patrons to order hot food, the act of walking into a restaurant can trigger the kitchen to start making an order.

Many other types of queue management exist, of course, and one of the advantages of building queue management with iBeacons is that the location of the queue entry point can move easily without reprogramming the application.

Bluetooth Low Energy Beacons

The key technology that enables the iBeacon is the proximity specification in Bluetooth Low Energy (BLE), which is often called Bluetooth Smart.[8] BLE is an enhancement of the existing Bluetooth specification designed for even lower-power operation. BLE beacons

8 For much more detail and information on Bluetooth Low Energy and its definition in the Bluetooth specification, see *Getting Started with Bluetooth Low Energy* by Townsend et al. (O'Reilly).

transmit identification information that applications can use to identify the type of space the beacon is installed in. Devices branded as "Bluetooth Smart Ready" are dual-mode devices that can interoperate with both classic Bluetooth as well as BLE.

iBeacons are a subset of the BLE beacon specification. Figure 1-1 illustrates the relationship between iBeacons, generic beacons, BLE beacons, and BLE devices. All iBeacons are BLE beacons, and all BLE beacons are BLE devices. However, there are beacons that are not Bluetooth-based, and there are BLE devices that do not beacon. This book concentrates on the overlap of all these categories, with minor notes where the iBeacon specification differs from generic BLE beacons.

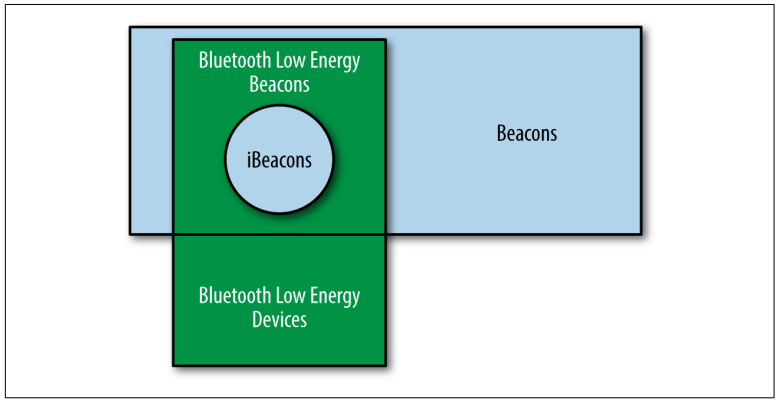

Figure 1-1. The relationship between iBeacons, BLE, and beacons

At low transmit power, the energy consumption of a beacon is quite low. Beacons can run for extended periods of time on battery power, with vendors claiming battery life of months or even years. Many proximity applications are designed to be triggered within a few feet of the target area and, in practice, that will require battery operation. In retail stores, for example, many displays will not have ready access to electrical power. Many early hardware products support limited battery lifetimes of a few weeks or months, so permanent displays will require full-time power.

Beacons also do not require Internet access to function. The protocol is simple and straightforward and can be embedded within an application so that it can run without continuous Internet connectivity. Note, however, that although iBeacons do not require

connectivity to function, the applications they enable typically do require connectivity to the device.

CHAPTER 2

The iBeacon Protocol

iBeacons are transmit-only devices. They work by periodically transmitting numerical identifiers that are mapped into actions by an application on a mobile device. As a result, the "protocol" that they use is simple and is basically a vehicle to deliver numerical identifiers to nearby clients.

Translating those numbers into concrete action requires building an app for mobile devices. The simplicity of the protocol hides the big challenge, which is that application developers and user experience designers need to figure out how to make applications interact more tightly with the physical world.

Bluetooth Basics of an iBeacon

The iBeacon advertising function is transmit-only. Many iBeacon devices will implement a Bluetooth receiver for monitoring and configuration functions, but the protocol itself is transmit-only.

In the jargon of the Bluetooth specification, an iBeacon is a *broadcaster*, a type of device that is specific to Bluetooth Low Energy. Broadcasters transmit periodic *advertising packets*, which contain information used by the receivers. iBeacon advertising packets are designed to be transmitted, but receivers do not need to respond to

them.[1] In effect, iBeacon advertising packets are thrown out into the air, and receivers can act on them (or not).

iBeacons send advertising packets. When a device receives an advertising packet, it results in the creation of an *advertisement event* at the receiver. Sometimes, advertising packets and events are also referred to generically as *advertisements*.

In effect, an iBeacon advertising packet says to the world, "Hello, I'm here, and my name is…" The difference is that the "name" in an iBeacon hello consists of three numerical identifiers:

Universal Unique Identifier (UUID) (128 bits)
: Roughly speaking, the UUID transmitted by an iBeacon is a 128-bit identifier that uniquely identifies the organization the beacon belongs to. When iBeacons are used by, say, a chain of stores, the UUID will indicate the beacon is run by the company.

Major number (16 bits)
: The Bluetooth and iBeacon specifications place no structure on the use of the major and minor numbers, but there is a hierarchy in the APIs used. The major number is used to identify major groups of beacons owned by one entity. In the example of a chain of stores, the major number will typically be used for all the beacons within one particular store.

Minor number (16 bits)
: The minor number is used to identify the lowest level of the hierarchy within a set of beacons. Returning to the example of a chain of stores, the minor number will be used for individual beacons within a single store location, perhaps to identify a product on display.

Proximity estimation uses the received signal power of a frame at the receiver, a number called the *received signal strength indication*

1 In the terminology of the Bluetooth specification, an iBeacon sends *nonconnectable undirected advertising packets*, which means that there is no attempt to establish any sort of connection between the iBeacon and any receivers.

(RSSI). RSSI is not transmitted in the advertising packet, because it is the power level of the signal when it reaches the receiver.

As far as data transmitted, the payload of an iBeacon consists of these three numbers. By far the most important, as well as most often misunderstood, is the UUID.

The Universal Unique Identifier

Many items in the Bluetooth specification use a UUID to identify unique elements. In iBeacons, the field called a UUID is most formally referred to as the *proximity UUID*, to distinguish it from any other UUIDs that might be in use by a Bluetooth device. The UUID is used to identify iBeacons that are under common management, and in effect, the UUID sits at the top level of the hierarchy of numbers.

Unlike other network protocols, such as 802.11, the UUID is not centrally managed to avoid conflicts. The Bluetooth protocol assumes that UUIDs are unique across all space and time. IEEE 802 networks, such as wireless LANs, have centralized assignment to guarantee uniqueness, but Bluetooth does not. With 128 bits to use in the identifier, it is likely that well-designed random number generators will choose unique numbers.

Most UUIDs are created by random number generators and will often incorporate the current time and an identifier of the generator (such as a MAC address). Many iBeacon configuration applications have a Generate button to generate a random UUID. Mac OS X has a command-line utility to generate UUIDs as well:

```
Matthew-Gasts-MacBook:~$ uuidgen
C743365A-C141-4C43-8A45-E258F7755D5B
```

Advertising Interval

Although the Bluetooth specification allows many advertising intervals, the iBeacon specification fixes the advertising interval at 100 ms. Setting the advertising interval is a balance between preserving battery life in beacons with long advertising intervals, but having

advertisements be frequent enough to enable services to be built with iBeacons.

Where to Get Specifications

Bluetooth specifications are published by the Bluetooth SIG and are freely available to the public. iBeacons are based on version 4.0 of the Bluetooth specification. The iBeacon specification is made available to Apple developers.

iBeacon Advertising Packet Contents

iBeacons transmit advertising packets, shown in Figure 2-1.

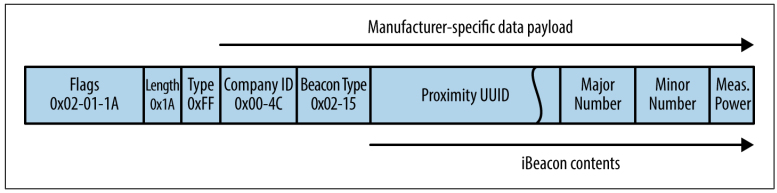

Figure 2-1. iBeacon advertising packet format

Advertising packets always have the same length and are composed of a series of fixed fields. The latter part of the frame contains manufacturer-specific information defined by Apple. The main reason for defining a custom packet format is to improve the ranging capabilities of an iBeacon.

Advertising Header

The iBeacon advertising header contains four fields, which, in practice, are fixed for iBeacons:

Flags (3 bytes, 0x02-01-1A)
: The Flags field is a length/type/value field. The first byte is a length indicator, which states that the Flags field has two additional bytes. The second byte describes the type, with a value of `1` indicating that the Value field contains flags. The Value field, which is the third byte, contains the flags themeselves. Most iBeacons report that they are capable of the general discovery mode and can use both low-energy and non-low-energy modes of operation. With those capabilities, the flag payload will be `1A`.

Length (1 byte, 0x1A, decimal 26)
: The remainder of the iBeacon advertising packet also has a length/type/value tuple. The Length field describes the length of the frame payload following this field. All iBeacon advertising packets are 30 bytes, but the Length field describes the length of the packet after three bytes of flags and the one-byte Length field for a total of 26.

Type (1 byte, 0xFF, decimal 256)
: This value indicates the contents of the frame are manufacturer-specific data. The first two bytes contain the company identifier, and the company is free to define how to interpret the remaining fields. In the case of an iBeacon, the Company ID will be set to Apple. Part of the reason for maintaining a licensing system for use of the protocol is that every iBeacon uses Apple's company ID, and Apple needs to ensure that devices implementing its ID are using it in the prescribed way.

Company ID (2 bytes, 0x004C)
: Technically, this field is the start of the manufacturer-specific advertising payload. The Bluetooth specification requires that manufacturer-specific payloads begin with the company ID number. iBeacons start with Apple's company ID number.[2]

As with most communications technologies, the header serves to introduce the payload, which immediately follows the end of the header.

iBeacon Payload

The payload is the workhorse of the iBeacon specification, and it contains the only fields in the entire advertisement that can be configured by the user. In addition to the three numerical fields, it also contains a calibration constant used in ranging.

These are the five fields in the iBeacon payload:

Beacon type (2 bytes, 0x02-15)
: Apple has assigned a value for proximity beacons, which is used by all iBeacons. Some sources state that this is a two-byte field,

2 The Bluetooth SIG provides a table of company identifiers within Bluetooth specifications.

with the first byte indicating a protocol identifier of 2 for iBeacon and the second byte indicating a length of 21 further bytes (15 in hex is 21 decimal).

Proximity UUID (16 bytes)
: This field contains the UUID for the iBeacon. Typically, this will be set to the organization that owns the beacon. Not all beacon products allow this field to be set.

Major (2 bytes) and Minor (2 byte) numbers
: These fields, each two bytes in length, contain the major or minor number that will be contained within the iBeacon's broadcast.

Measured power (1 byte)
: Implicit within the iBeacon protocol is the idea of *ranging* (identifying the distance a device is from a beacon, as discussed in "Ranging" on page 48). There may be slight variations in transmitter power, so an iBeacon is calibrated with a reference client. Measured power is set by holding a receiver one meter from the beacon and finding an average received signal strength. This field holds the measured power as a two's complement.[3] For example, a value of C5 indicates a measured power at one meter of –59 dBm.

Distance from an iBeacon is calculated by estimating the distance a signal must travel for it to fade to the level at which it is received, as adjusted by the calibration constant.

Although the iBeacon frame format is quite simple and has only a few variable parameters, those parameters are enough to support complex applications, as you will see in the next few chapters.

3 Two's complement is a common way of storing integers in computing.

CHAPTER 3

Setting Up Your Own Beacons

There are many ways to get a beacon running. With such a simple protocol, all that a beacon needs to do is transmit its UUID, major number, and minor number at a fixed interval. Any device with Bluetooth 4.0 (or later) hardware is capable of acting as a beacon, whether it is a software application on a laptop, a software app on a mobile device, or a host computer with a USB interface.

Types of Hardware

Creating a beacon does not require any specialized type of hardware. You might already have suitable hardware lying around to create a beacon. If not, the investment required is only a few dollars.

Dedicated Beacon Hardware

The advantage of dedicated beacon hardware is that it is cheaper than using general-purpose hardware, and it is optimized for the beacon task. Here are a couple popular options:

Estimote
: One of the earliest developers of beacon technology, Estimote sells a developer kit that includes three beacons for $99. Interestingly, Estimote beacons have fixed configuration parameters and, in particular, administrators cannot set the UUID.

RadBeacon
: RadBeacon is a $29 USB dongle that performs the transmission functions of an iBeacon. All you have to supply is USB power. Configuration of the beacon's numbers is done through an app.

Kontakt
: Kontakt sells an ARM-based iBeacon as well as tools for managing iBeacons and analyzing user interactions with them.

Gelo
: Gelo's Beacons are waterproof and designed for both indoor and outdoor use, and the batteries can be replaced manually using simple tools.

Some dedicated iBeacon hardware runs on batteries, either coin cell or something with a higher capacity, such as AA.[1] The iBeacon protocol is simple, and the hardware was designed to run on small batteries for extended periods of time.

One of the major reasons for designing such a simple protocol is that it allows beacons to be made cheaply. Additionally, such a simple protocol can run on battery power for extended lengths of time, which enables proximity applications to be developed for areas that might otherwise be inaccessible if beacons were required to be connected to a higher-power source.

Within Apple's iBeacon specification, the advertisement interval is fixed at 100 ms. That is, an iBeacon will always transmit 10 times per second. This decision embodies a trade-off. Frequent beacon advertisements enable more immediate actions to be taken by an app, while extended advertisement intervals prolong battery life. The iBeacon specification requires all devices to adhere to the same advertisement interval, regardless of how the beacon is being used.

The specification's requirement for a relatively high fixed transmit interval ensures that applications will react promptly, because beacon transmissions will be readily available. It also will enable iBeacon-powered applications on enduser mobile devices to save power, because the search for beacon advertisements can be much shorter. The downside to a relatively high advertisement interval is

1 A January 2014 teardown of the Estimote beacon in *Make* magazine noted that approximately a fifth of the battery capacity was consumed in a month. Tests since seem to indicate that battery life claims are often optimistic.

reduced battery life. If iBeacons are battery powered, the operational lifetime between battery changes may be quite short.

General-Purpose Hardware

Rather than use specialized hardware, beacons can be generated using general-purpose hardware, such as the following examples:

Macs
: Newer Mac laptops and desktops include BLE chipsets and can act as an iBeacon. iBeacon capability came to the MacBook Air in 2011, the iMac and MacBook Pro in 2012, and the Mac Pro in 2013. When using OS X, a program is needed to program the BLE chipset to act as an iBeacon, such as the $10 MacBeacon program.

iOS devices
: Newer iOS devices (running iOS 7)—including iPhone 4S or later, iPad 3 or later, iPad mini, and fifth generation iPod touch—can act as beacons by using one of the many free programs that use the BLE hardware through the appropriate development frameworks.

Raspberry Pi
: The Raspberry Pi ($15 for USB 4.0 interface, plus the Raspberry Pi) is a tiny Linux machine, capable of running a Bluetooth stack. With the addition of a Bluetooth 4.0 USB adapter, a Raspberry Pi can act as an iBeacon.

Arduino
: Kytelabs BLEduino, a Kickstarter-funded project, promises to add iBeacon functionality to the Arduino world when the project ships. The RedBearLab BLE Mini can be used to add iBeacon functions to development platforms with a serial interface.

Nordic Semiconductor
: The Nordic Semi nRF1822 is a tiny 20 mm diameter iBeacon reference design.

These options often provide a useful way to test the interaction of apps with a beacon, because they can be used for proof-of-concept work.

Activating a Beacon

Once you have identified the supporting hardware, making it act as an iBeacon is straightforward.

iOS Devices

Newer iOS devices (see "General-Purpose Hardware" on page 21) have the ability to use the built-in BLE hardware to become an iBeacon. With the radio hardware and a stack, any application can activate iBeacon capabilities to add proximity information. For example, a kiosk application could act as an iBeacon, automatically signing in end users who come into the vicinity of the kiosk.

If all you want is a test iBeacon, though, many applications can offer that functionality. The one I use most is Locate Beacon by Radius Networks, available free from the App Store.

After starting up the application, tap the iBeacon Transmitter button to bring up a list of configured iBeacon transmitters, as shown in Figure 3-1. One configuration can be set for transmission, which is indicated with a green lightning bolt to the left of the name.

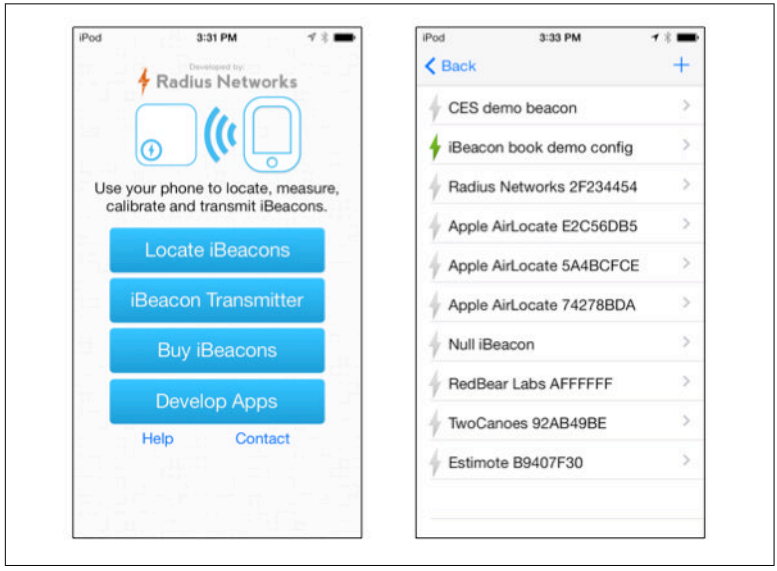

Figure 3-1. Locate Beacon's main screen and transmit list

To add a new iBeacon configuration, tap the plus sign in the upper-right corner. You will be taken to the iBeacon transmitter configuration screen shown in Figure 3-2.

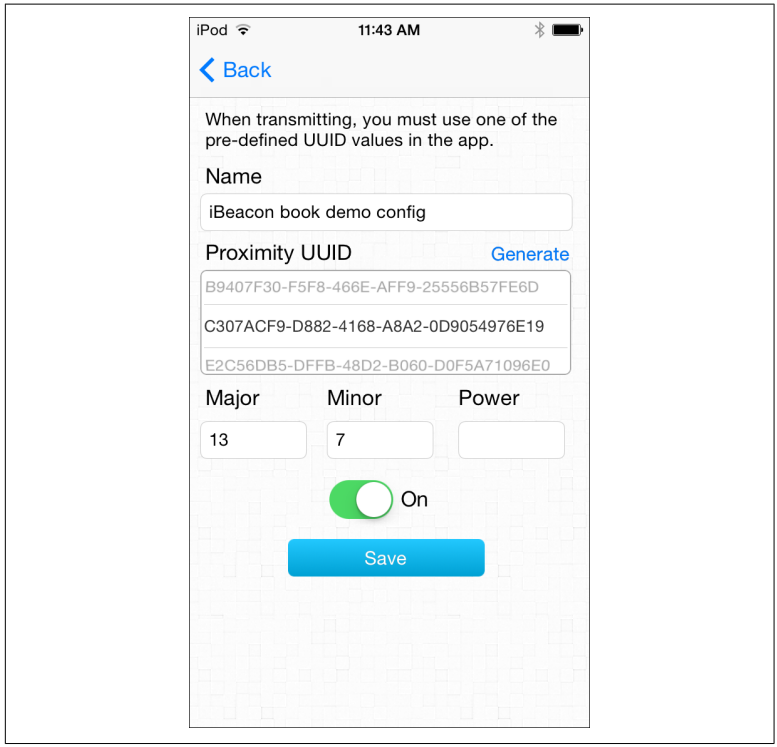

Figure 3-2. Locate Beacon transmitter configuration

The elements on the transmitter screen are straightforward:

Name
: This is a string identifier held within the application. It has no effect on any parameters transmitted on the BLE interface.

UUID
: Every iBeacon must have at least a UUID. If you are configuring a specific UUID, it can be typed into this field. If you are generating a first-time iBeacon for use with an application, use the Generate link above the UUID field to generate a random UUID for use with your application.

Major and minor numbers
: These are the major and minor numbers transmitted in the iBeacon advertisement packets. As a two-byte number, the range must be between 0 and 65,535. If they are left blank, a zero will be transmitted.

Power
: The value of this field ranges between –1 and –255, representing the measured power field in the iBeacon. If left blank, it will default to –59 dBm.

Transmitter switch
: By default, a newly configured iBeacon will be off. If you switch on the configuration, the iOS device will act as a beacon for the current configuration only and will switch from any active configuration.

Not surprisingly, the controls on the application correspond with the mutable fields in the iBeacon protocol, plus a switch to trigger transmission. With a simple protocol, a one-screen configuration is easy to achieve.

RadBeacon

The Radius Networks RadBeacon is a USB-powered iBeacon that is configured over Bluetooth. The advantage of USB power is that power adapters are cheap and plentiful, and an iBeacon can be placed anywhere that USB power is available. It is a good test device for demonstration purposes or for work with application development, because it is inexpensive and most people will have a USB charger lying around.

Figure 3-3 shows a RadBeacon inserted into an Apple USB power adapter. I often store my RadBeacon attached to power, because it is easy to remove the RadBeacon if I need power.

To configure the RadBeacon, install the free RadBeacon app from the App Store on an iOS device. The app will activate and look for RadBeacons in its vicinity. From the list, you can drill into any single RadBeacon to set its configuration.

You can change the parameters of RadBeacons directly from the app using the screen shown in Figure 3-4. The main configuration will of course be the numerical identifiers placed in the advertising packets.

Figure 3-3. A RadBeacon plugged into a USB power adapter

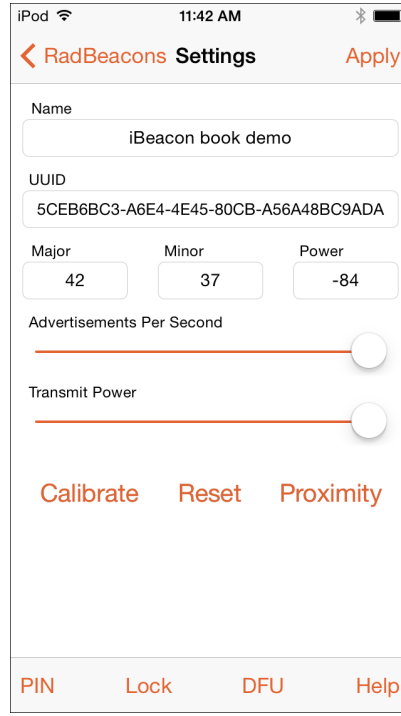

Figure 3-4. RadBeacon parameters

For security reasons, a RadBeacon can have its configuration changed only within the first 30 minutes after powering up. After that, you must turn the power off and turn it back on, which can be

as simple as unplugging the RadBeacon from its USB host to cause it to reboot.

Raspberry Pi

The Raspberry Pi has taken the world by storm as a cheap prototyping platform because it has relatively high capabilities, plus the programmability of Linux, at a price that would have been unthinkably low even just a few years ago.

Raspberry Pi iBeacon hardware

It doesn't take much to set up a Pi as an iBeacon. The hardware shopping list has only four items, and even if you are starting from scratch, you probably have one or two of them lying around:

Raspberry Pi (about $35)
: Either a Model A or a Model B will work. I use a Model B, because it has a built-in Ethernet port and two USB ports. You will be using one USB port for the Bluetooth adapter, so if you also want to have a Wi-Fi interface, you will need to have the Model B with its two USB ports.

Bluetooth interface (about $15)
: The IOGEAR GBU521 is a common choice, because it is affordable and uses one of the major Bluetooth chips on the market: the Broadcom BCM20702A0 chipset. This adapter supports both classic Bluetooth as well as BLE; setting it up as an iBeacon will use only the BLE mode. When purchasing a Bluetooth adapter, ensure that it is a Bluetooth 4.0 adapter. Earlier Bluetooth adapters are still readily available, but they do not support iBeacon functions.

SD card
: The SD card will act as the disk for the Linux installation. It must be at least 4 GB, but much larger disks are still quite inexpensive. When I assembled my first Raspberry Pi iBeacon, a 16 GB card was around $10.

A micro-USB power adapter (about $15)
: To run multiple USB devices, get a high-current adapter capable of supplying 2 amps.

In addition to the Raspberry Pi, you will want the following tools:

- USB keyboard and mouse and a monitor with HDMI for the initial configuration (in a pinch, a television with an HDMI input can serve as the monitor)
- HDMI cable
- Software tool (as described in "Installation and setup" on page 27) to watch for iBeacons to verify that it is working

Once you have everything in one place, it is time to get started. Setting up a Raspberry Pi is simple, but there are enough steps that I would budget an hour or two for the entire process.

Installation and setup

Several forms of Linux can run on Raspberry Pi. This example uses the Raspbian distribution, which is available from the Raspberry Pi support site.

The core software required to set up an iBeacon is the BlueZ stack, which added support for Bluetooth Low Energy in the 5.x series. BlueZ is not part of the Raspbian distribution and needs to be compiled before it will run.

The process for getting the Pi running as an iBeacon is straightforward:

1. Download Raspbian (or your preferred Linux distribution).
2. Write Linux onto a digital memory card. This process varies depending on the operating system of the computer that is creating the memory card.
3. Boot your Pi and perform an initial configuration. For the purpose of starting up an iBeacon, the main configuration you'll probably want to do is to enable SSH access. If the Pi is connecting to the Internet with Wi-Fi, the wireless interface will need to be configured, too.
4. Update Raspbian. Although Raspbian is released frequently, I usually begin by updating to the most recent software available by using the package system:

```
mmmpi$ sudo apt-get update
mmmpi$ sudo apt-get upgrade
```

5. Install the support libraries. Several USB libraries are required to run Bluetooth interfaces, and they can all be installed with

one command. The downloads are not substantial and should take only a few minutes:

```
mmmpi$ sudo apt-get install libusb-dev libdbus-1-dev
libglib2.0-dev libudev-dev libical-dev libreadline-dev
```

6. Download and install the BlueZ stack to control the Bluetooth interface:

```
mmmpi$ cd bluez-5.18
mmmpi$ sudo ./configure --disable-systemd
mmmpi$ sudo make
mmmpi$ sudo make install
```

At the time this book was written, the version of BlueZ in the package system was quite old. You need to install version 5 or later, so you might need to compile it.

7. Shut down the Pi, insert the dongle, and restart. Once it reboots, check if the device is discoverable, both by looking at the USB devices and by running `hciconfig`. The IOGEAR module uses a Broadcom chip and appears last:

```
mmmpi$ lsusb
Bus 001 Device 002: ID 0424:9512 Standard Microsystems
Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root
hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems
Corp.
Bus 001 Device 004: ID 0a5c:21e8 Broadcom Corp.
mmmpi$ hciconfig
hci0: Type: BR/EDR  Bus: USB
       BD Address:00:02:72:C8:B2:EB ACL MTU:1021:8 SCO MTU:
64:1
      DOWN
      RX bytes:547 acl:0 sco:0 events:27 errors:0
      TX bytes:384 acl:0 sco:0 commands:27 errors:0
```

8. Bring up the Bluetooth interface and configure the iBeacon data. Make sure scanning is turned off before configuring the beacon. To check, run `hciconfig` and make sure the `SCAN` flag is not present.

   CLI tools can be used to configure iBeacon data, which must be in hex format. The beacon data must be entered directly as a string of numbers, as shown in the following command:

```
mmmpi$ hciconfig hci0 up
mmmpi$ sudo hcitool -i hci0 cmd 0x08 0x0008 1e 02 01 1a 1a
ff 4c 00 02 15 e2 c5 6d b5 df fb 48 d2 b0 60 d0 f5 a7 10 96
e0 00 00 00 00 c5 00 00 00 00 00 00 00 00 00 00 00 00
< HCI Command: ogf 0x08, ocf 0x0008, plen 44
  1E 02 01 1A 1A FF 4C 00 02 15 E2 C5 6D B5 DF FB 48 D2 B0
60
  D0 F5 A7 10 96 E0 00 00 00 00 C9 00 00 00 00 00 00 00 00
00
  00 00 00 00
> HCI Event: 0x0e plen 4
  01 08 20 00
```

9. Finally, turn on the Bluetooth advertising capability to begin sending out the iBeacon data:

   ```
   mmmpi$ sudo hciconfig hci0 leadv 3
   ```

   Once again, the `hciconfig` command is used to control the interface. The command uses `leadv` to activate low-energy advertising, and the number `3` to indicate nonconnectable advertising, which is what is required by the specification.

With the last step complete, you now have a functioning iBeacon that can be used to trigger an application to take actions.

CHAPTER 4
Application Development

iBeacons and the applications they empower are tied at the hip. One of the benefits of using such a simple protocol is that it is limited only by the imagination of the developer community.[1] This chapter describes the basic elements of application development with iBeacon—in essence, the seeds from which all iBeacon-enabled applications grow.

iBeacons have one operation: to transmit advertisement packets. Mobile devices either receive the beacon transmission or do not. Everything in the iBeacon ecosystem flows from that simple statement. Proximity is defined as "I am close enough to the beacon to receive its transmissions."

As discussed in Chapter 2, iBeacons repeatedly transmit three numbers, and that's it. Those numbers are used to trigger actions on the device, but otherwise, a beacon transmission is not all that interesting. If an application developer wants to interact with the user by popping up a message, contacting a server, or pushing data to the user, that must be done by programming an application to take those actions.

A beacon's advertisement includes no descriptive text or mapping. Translation between the beacon's transmissions and any actions are done entirely within an application on the receiver, even if the

1 One of the many organized efforts to quickly build applications with iBeacon is the February 2014 hackathon held in San Francisco: Bring Home the Beacon Hackathon.

application is a simple text message to say, "Welcome to the region." iBeacons are not servers, and they do not need to have storage for anything beyond the contents of the numbers in the advertisement packet.[2]

To add interactivity based on an iBeacon, an application needs to query its own local storage or a web service to learn more. By itself, an iBeacon is not capable of anything beyond simple broadcast messages.

Limitations on iBeacons

The main reason for requiring an application is that iBeacons don't receive or process any information.[3] As a result, while beacon hardware can be made incredibly simple, conversely, varying levels of complexity must be put into the companion application.

As a transmit-only device, an iBeacon cannot perform an action that depends on receiving data from a device. iBeacons cannot be directly responsible for either of the following:[4]

Reporting on nearby clients
: Beacons are transmitted in only one direction, with no response from mobile devices. Without any handshaking or reception built into the protocol, there is no way for an iBeacon to learn about devices in its immediate area.

Enabling mobile devices to learn about other devices
: iBeacons cannot directly assist in establishing device-to-device communication. iBeacons are limited by the transmit-only protocol; a receiver cannot determine from the contents of a beacon transmission whether it is the only receiver of that beacon or one of several hundred. To establish peer-to-peer

2 Many iBeacon devices have more feature-rich capabilities, such as flash memory to store upgradeable firmware and a configuration server.

3 Many iBeacon devices use two-way Bluetooth interfaces to receive configuration. However, the beacon functions do not use reception.

4 Many of these actions can be accomplished by applications running on mobile devices. It is important to make the distinction between the iBeacon protocol and what an entire iBeacon-based system can accomplish.

connections between devices, bidirectional communications are needed, and iBeacon is unable to act as a broker.

iBeacons cannot directly interact with the mobile device and, at best, they are triggers for other applications. However, due to the lack of direct interaction and the simplicity of the protocol, an iBeacon also cannot:

Send a message to a mobile device
: An iBeacon's transmission consists of three numbers to uniquely identify what the device is near. To translate those three numbers into an action or a message, an application needs to be involved. That application can be as simple as "when the UUID, major number, and minor numbers match, display the string: *welcome to store #123*."

Get access to latitude and longitude information
: iBeacons transmit identification numbers, not a geographic location. In order to get latitude and longitude, an app would need to either use a technology like GPS or translate an iBeacon's numerical identifiers into a geographic location using a mapping database.

Anything that an iBeacon can do is accomplished by a few simple protocol operations, plus a dash of logic in an application. Although an iBeacon cannot always directly provide support to a mobile device, it often enables an application to do so.

Basic iBeacon Programming Functions

iBeacons enable location to be reported to an application through two operations:

Monitoring
: A high-level view of whether a device is within range of a specified beacon. It indicates that both the beacon and the mobile device are loosely within the same space, as defined by whatever the transmission characteristics of the beacon are.

Ranging
: Specific to a single beacon, ranging uses its transmissions to estimate the distance from a mobile device to a beacon.

Monitoring and ranging are related to the position of a mobile device relative to an iBeacon, but they each convey slightly different information.

Monitoring

An iBeacon's transmissions define physical space, often called a *region*. A device is either "in" the region and able to receive and decode beacon transmissions, or it is "out" of the region and unable to receive transmissions. In essence, monitoring tells the application whether the device is close to a point or a resource of some sort.

Monitoring is a one-way function: a device listens for beacons. The beacons themselves cannot tell a device what is around, though of course they can report information to an app that can.

Monitoring can be carried out on any stage of the numerical hierarchy in the beacon protocol:

Match UUID only
: Any beacon matching the UUID will be reported to the application. Matching on the UUID ensures only the broadest possible set of triggers, because any iBeacon associated with the application will trigger action. In the case of a store chain, monitoring for UUID will wake up the application only when the device enters a given store.

Match UUID and major number
: Rather than match any possible iBeacon, applications can instead choose to monitor for a particular subset of iBeacons collected under one major number. In the case of a store chain, the application developer might want to identify the device entering a particular store.

Match UUID, major number, and minor number
: When all three numbers are matched, it is likely that the application is targeting a particular iBeacon, perhaps a display within a single store.

Operating systems have the ability to monitor for multiple iBeacons at a given time, and applications might change the monitoring list as circumstances change. For example, an app that deals with a chain

store will probably begin by monitoring on UUID only to determine when the device is near any of the chain's locations. Once monitoring activities determine that the device is near a location, the application can read the major number and set up a new monitoring trigger to look for anything in the store, or even particular displays within the store.

An application can change the regions it monitors for dynamically, looking at wider and narrower scopes, depending on where the device is in relation to what is around it.

Although the iBeacon specifications allow configuration of any number of monitoring rules, many operating systems limit the number of simultaneous regions that can be added to the monitoring list. This could be because if multiple applications are running, it is possible to fill up the monitoring list with ease. The size of the monitoring list is a decision made by the operating system developer and is not a fundamental limitation in the protocol. At this point, however, it is important to always check the return code when you begin monitoring to ensure that it was successful.

Most operating systems have a relatively small limit for the number of regions that can be monitored.

Monitoring rules communicate to an application when a device is either in range of an iBeacon or set of iBeacons, or when the device has moved out of range. Wireless signal strength can be jumpy and shift dramatically, even when the transmitter and receiver remain in the same physical space. Therefore, many iBeacon receivers will implement *hysteresis* to smooth out (or *calm*) monitoring alerts. Notifications that a device has entered or left an area will happen within a few seconds, but they are not guaranteed to be immediate.

Ranging

The second major operation performed by mobile devices is *ranging*: determining how far away a device is from a particular iBeacon. Based on the transmission strength and calibration constant

transmitted in the beacon, the receiver can use the received signal power to estimate distance.

Measuring the range is a relatively high-power operation, because the device must listen for and process all beacon advertisements, requiring significantly more activity with its radio.

A common use of ranging operations is to determine which iBeacon is closest to a receiver. For example, a museum exhibit hall might contain several significant displays, all with an iBeacon. A device might use ranging to determine which exhibit is closest to the device, so that the user interface may sort retrieved data by distance to the exhibit. However, it is bad interface design to assume that the user is always interested in the closest exhibit.[5]

When there are multiple iBeacons near a device, do not assume the user is interested in the closest one. Instead, offer a choice between the nearby iBeacons.

Calibration and Ranging Accuracy

Accurate ranging information depends on correctly setting the calibration constant, as described in the "Measured Power" entry in "iBeacon Payload" on page 17. A mobile device uses the received signal power, as adjusted by the calibration constant, to calculate the range.

Calibration assumes that an iBeacon is placed in free space, and the major factor in signal loss will be the inverse-square fading of the signal. In practice, the reported range often bounces around because of *multipath interference*. When a radio signal can take multiple paths between a transmitter and receiver, radio energy from each path can interfere with the other paths. In some situations, the interference is constructive and results in a stronger signal; in others, the interference is destructive and partially cancels out the transmission. Even though the physical environment is

5 As an extreme example, consider Paolo Veronese's massive The Wedding Feast at Cana, a one-and-a-half-ton masterwork with an area over 720 square feet (67 square meters). To appreciate such a large painting, a museum patron might move back to take in the full view and be closer to an iBeacon on another exhibit in the same room.

fixed, multipath interference may cause the received signal strength to fluctuate rapidly. Ranging algorithms must be aware of the potential for multipath interference and smooth out variations if the range is used by a user-facing function.

Advanced iBeacon Programming Functions

iBeacons trigger applications on the device to take some action. Those actions might be based on data stored locally on the device, or they might trigger the device to interact with network-based services. Once an iBeacon is detected, it is up to the application on the device to determine what, if anything, is needed from elsewhere on the network.

Web Services Interactions

For interactivity, iBeacon-powered applications need to translate the tuple of numbers in the iBeacon advertisement into something *actionable*. Two approaches are possible, and the one you choose depends on the amount and stability of the information:

Local storage
: When an iBeacon advertisement is received, the application refers to local storage to look up the beacon parameters. This approach is best when the application must operate without network connectivity and when the information is long-lived, because it requires users to update the application (or at the very least, a beacon database) to change the application behavior. The application will need adequate storage for the beacon database, which might be a consideration if the number of iBeacon identifiers is quite large.

Web service
: As an alternative, the application can look up the iBeacon in a database and retrieve information about the beacon for use by the application. If the number of beacons is large, or if they change position or meaning frequently, this approach works better. From a practical perspective, an application that uses real-time status information will need to use a network service to hold device location information, for example.

Whether you choose for an application to use local storage or a web service, interacting with external data stores is an important method of adding interactivity.

Geofencing and Location Services

One of the initial applications of iBeacons is to perform *geofencing*, where services are restricted based on proximity. For example, a device can report that it is inside a building and receive a higher level of service.

As another example, iBeacon transmissions are low power and, practically speaking, can be readily confined to a room. Restricing service visibility to areas where a beacon is visible can assist users in choosing from a long list of network services.

Security and Privacy

With a relatively simple protocol, iBeacons have straightforward privacy and security implications. Essentially, the protocol provides very little in the way of management tools, and there are huge opportunities to add on to the basic framework.

Basic iBeacon Security

The main security threat to iBeacons is that the protocol provides no cryptographic security at all. The contents of the beacon frame are not encrypted or authenticated, and are therefore quite easy to spoof. It is not difficult to learn the numeric identifiers from an iBeacon and assign them to a pirate beacon.[6]

With essentially zero security in the protocol, applications have to provide their own defenses against spoofing. Most importantly, applications should be designed so that the mere presence of an iBeacon does not trigger an action with security consequences.

Here are a couple approaches for designing defenses into an application:

6 For example, one of the first "treasure hunt" applications was shown at CES in January 2014, and it was promptly reverse engineered by Alasdair Allan and Sandeep Mistry.

Timestamping
: When an application receives a beacon transmission, record it on a network service that can provide a trusted timestamp. Software can then analyze a beacon track to ensure that it was completed in a reasonable amount of time. For example, a major convention show floor takes several minutes to walk across, so rapidly programming a beacon to spoof several transmitters will show an impossibly fast timestamp trace.

Cross-references
: Cross-reference iBeacons with some other source of information. When a beacon transmission is received, cross-reference against other location attributes to ensure that the receiving device was in the right neighborhood. Does the GPS lock match the expected location,[7] and are Wi-Fi access points in the area the access points that are around the "real" beacon?

Use an iBeacon to trigger some other sort of transaction on the network, and protect that transaction with appropriate security techniques such as Transport Layer Security (TLS), the protocol that protects nearly all web transactions.

Most importantly, ensure in the application design that the presence of an iBeacon itself is not the sole security feature of an application. The best design is to use the iBeacon to kick off transactions, but to ensure that any monetary or personally identifiable transaction is protected by a strong cryptographic channel.

Privacy

iBeacons themselves do not have significant privacy implications, because they cannot receive signals from a mobile device. However, the presence of iBeacons in a physical space can readily enable tracking because an application can record a *beacon track* and report it to a network service for further analysis.

7 The Radius Networks Proximity Kit can perform some of these functions automatically and transparently within your app.

Be aware that just because an iBeacon itself does not track users, that does not preclude the iBeacon from being an integral part of a user-tracking system.

Location information is already subject to user control on many mobile devices, whether at install time or at runtime. Ideally, an opt-in privacy framework would include a specific setting for beacon-based or Bluetooth-based location information; unfortunately, that level of granularity is not yet supported.

As you develop an application, be clear about the benefit to the end user for sharing location information. Users are likely to accept indoor navigation and help locating hard-to-find items much more readily than turning their mobile devices into yet another channel for advertisements.

When writing an application, be careful about collecting tracking information without explicit user authorization.

Over time, it seems likely that mobile ecosystems will offer better control of iBeacon information for end users, potentially even including standard practices for approved applications in the official app stores.

CHAPTER 5
iOS and iBeacons

This chapter describes how to make iOS devices work with iBeacons. In keeping with the simple protocol, the APIs that serve iBeacon applications are also quite simple, though deceptively so. APIs provide support for the major protocol operations described in the previous two chapters: understanding when a device is near iBeacons, determining the identity of those iBeacons, and recognizing when the number and identity of iBeacons changes. It is also possible to use built-in features of iOS to interact with iBeacons, most notably, Passbook passes.

iBeacon Development on iOS with Core Location

Most of the iBeacon functions supported on iOS are provided through the iOS *Core Location framework*.[1] Core Location has long enabled devices to use physical location attributes in an application, and iBeacon proximity functions were added in iOS 7.0.[2]

1 For the purposes of this book, the terms *framework* and *API* are used interchangeably. Formally, a framework consists of libraries and support code, in addition to the interface, and is a superset of an API, which is the public interface to use functionality exposed by the framework.

2 The iOS Core Location developer documentation describes both the location functions and the new proximity functions. For more information on developing for iBeacons, look at the beacon classes.

Users allow applications to make use of the Core Location framework. In the Privacy panel within iOS, shown in Figure 5-1, the device's owner can block all applications from using Core Location functions, including iBeacon support, or selectively block applications from using Core Location. Naturally, devices must have hardware support for the underlying Bluetooth Low Energy functions, which in practice means an iPhone 4S or later iOS device. Macs sold since late 2011 also have the required Bluetooth hardware.

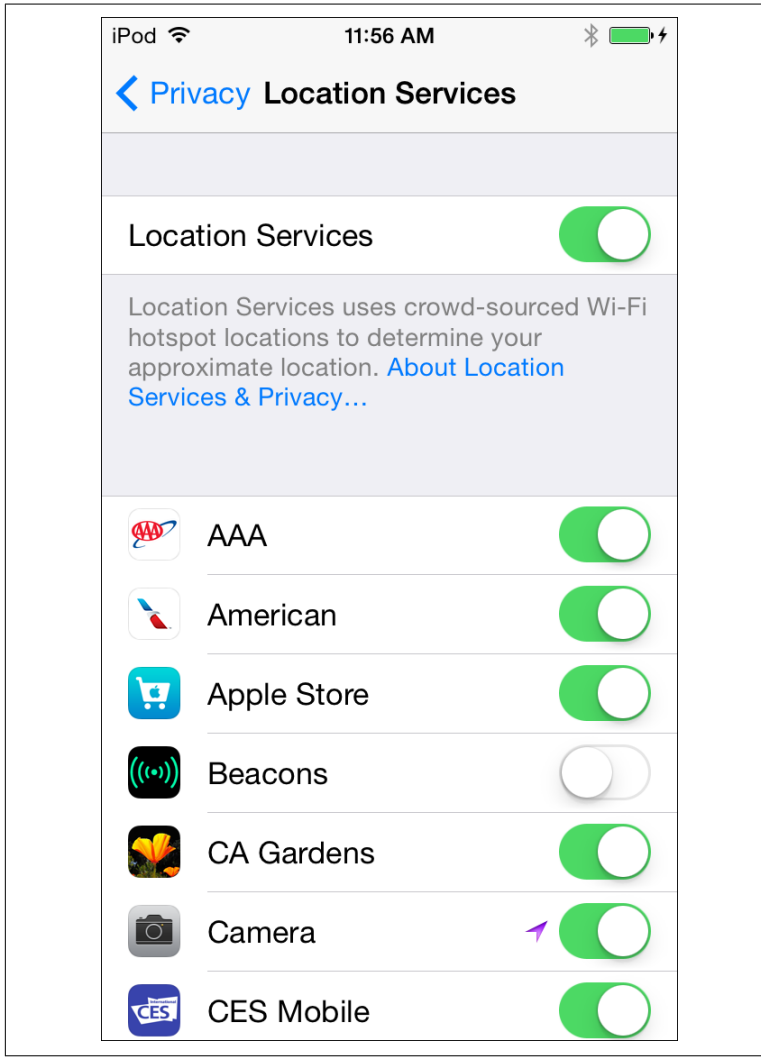

Figure 5-1. iOS Location Services control panel

iBeacon support is provided only in iOS 7 and later.

Core Location provides the conceptual bridge between a beacon's radio transmissions in the physical world and actions taken on a device to interact with the world. The core capabilities offered by the programming API are straightforward: notifying when a device has entered or left the area defined by a beacon's transmission, and estimating the distance between a device and a beacon.

iBeacon Monitoring with Core Location

Monitoring, performed in both the foreground and the background on iOS, is used to determine when a device has entered or left an iBeacon's coverage area. In the context of the framework, an iBeacon defines a *region* through its transmissions. In Core Location, a region is the space in which a specified combination of UUID, major number, and minor number is received. Devices are in the region when they receive and are able to decode the beacon and the numerical identifiers match.

When a region is set up for monitoring purposes, the framework requires that it be identified. Identification is, not surprisingly, accomplished by matching the numbers that a beacon transmits. The framework imposes a hierarchy and permits only three types of filter:

UUID only
: Any time an iBeacon's transmissions match the UUID, report the match. In this case, a region might be quite large and consist of several iBeacons, because anything with the UUID will match regardless of major or minor number. In practice, this likely corresponds to any iBeacon installed in support of an application anywhere in the universe.

UUID plus major number
: This option narrows the region by requiring a match of both the UUID and major numbers. Like the UUID-only option, it is likely to match several iBeacons, most likely installed at one particular location.

UUID plus major and minor numbers
: The most specific of the three options, this region type requires a match between the UUID, major number, and minor number. In most cases, this option will match only one specific iBeacon.

Pseudocode for setting up iBeacon monitoring is simple:

1. Create a `CLBeaconRegion` object.[3] This object is defined by the UUID, major number, and minor number. In the following code, the iBeacon to monitor has the UUID `24F7B8B3-D124-4B7B-A180-CBA317CC1BB6` (generated specifically for this book by the `uuidgen` tool) and has the major and minor numbers of `42` and `105`, respectively. To monitor more generically, create the `CLBeaconRegion` object with only the numerical parameters desired—for example, only the UUID.
2. Create a `CLLocationManager` to monitor for that region to obtain notifications when the device crosses region boundaries.
3. Take action based on the region boundary crossings. These actions are developer-defined and likely involve interaction with something else to make the application respond or react. It can be as simple as displaying a message, but the type of action is limited only by the developer's imagination.

Translating the pseudocode into source code yields something like the following:

```
NSUUID *myUUID =
   [[NSUUID alloc]
   initWithUUIDString:@"24F7B8B3-D124-4B7B-A180-CBA317CC1BB6"];
CLBeaconMajorValue myMajor = 42;
CLBeaconMinorValue myMinor = 105;

// This region matches on anything with the UUID
CLBeaconRegion *uuidMatch = [[CLBeaconRegion alloc]
               initWithProximityUUID:myUUID
               identifier:@"uuidMatch" ];

// This region matches on one specific beacon in the UUID
CLBeaconRegion *matchThree = [[CLBeaconRegion alloc]
               initWithProximityUUID:myUUID
               major:myMajor
               minor:myMinor
```

3 For full details on `CLBeaconRegion`, see the Apple developer documentation.

```
                    identifier:@"matchThree" ];

    CLLocationManager *myLocationManager =
                        [[CLLocationManager alloc] init];

    // Check that hardware supports monitoring
    if ([CLLocationManager
         isMonitoringAvailableForClass:[CLBeaconRegion class]])
    {
      // Monitor for a UUID
      [myLocationManager startMonitoringForRegion:uuidMatch];

      // Change to monitor for a specific iBeacon within that UUID
      [myLocationManager stopMonitoringForRegion:uuidMatch];
      [myLocationManager startMonitoringForRegion:matchThree];
    }
```

This sample code defines the three numerical values for the region and then defines two regions based on those values. The first region monitors only for a UUID, while the second region monitors for an iBeacon that uses all three numbers.

Monitoring for the region is simple. Call a `CLLocationManager` to start monitoring. To change from monitoring the UUID-only region to look for a specific trio of numbers, stop monitoring and restart it with a more restrictive parameter set.

In practice, before passing a region to a location manager, check to see that it actually implements monitoring. Not all iOS devices have iBeacon-compatible Bluetooth hardware, and users can disable iBeacon functions in the privacy panel. Call the `isMonitoringAvailableForClass` method to determine if monitoring is available. In the sample code above, the function is used to determine that iBeacon functions are available before proceeding with calls to the monitoring methods.

Taking Action

Noticing that a device has crossed a region boundary is the first step in taking action. Upon entering a region, one common action to take might be to begin ranging, as described in "Ranging" on page 48.

The operating system delivers region events in three ways. Location managers can include one of the following delegated methods when events occur:

- The location manager can define a method for `didEnterRegion`, which is called when a device crosses the boundary to enter a region.
- The location manager can define a method for `didExitRegion`, which is called when a device crosses the boundary to leave a region.
- If the `requestStateForRegion` method is called, region-crossing events will be delivered as they occur and will call the `didDetermineState` method.

An application can take any action triggered by the program. Commonly, the application will display a notification, though it could also lock the screen or trigger a web service call to proceed further. Applications have a few seconds to execute these tasks.

Continuing our example, the application might need to first determine whether it is within the coverage area of a given UUID and then try to lock on to a particular beacon. Naturally, the code could also begin ranging beacons or could even change the monitoring list to look for a specific beacon.

This sample code displays a simple message to indicate the motion relative to an iBeacon's region:

```
// Delegated location manager method called when
// iBeacon region is entered
- (void)locationManager:(CLLocationManager *)manager
                        didEnterRegion:(CLRegion *)region {

    // Tell the user the device has entered the region
    UILocalNotification *message =
                        [[UILocalNotification alloc] init];
    message.alertBody = @"You are here, and I am happy!";
    message.soundName = @"Default";
    [[UIApplication sharedApplication]
       scheduleLocalNotification:message];
}

// Delegated location manager method called when the
// device departs the region
- (void)locationManager:(CLLocationManager *)manager
                        didExitRegion:(CLRegion *)region {

    // Tell the user the device has left the region
    UILocalNotification *message =
                        [[UILocalNotification alloc] init];
```

```
        message.alertBody = @"Leaving beacons makes me sad."
        message.soundName = @"Default";
        [[UIApplication sharedApplication]
           scheduleLocalNotification:message];
    }
```

Using `didEnterRegion` and `didExitRegion` requires writing and maintaining two methods. The operating system can also pass the programmer a list of region entry and exit events and use one method to take action.

By using `didDetermineState`, a programmer can combine the two actions together in one method, though it will probably be necessary to use the `CLRegionStateInside` variable to determine if a device is inside or outside the region:

```
- (void)locationManager:(CLLocationManager *)manager
          didDetermineState:(CLRegionState)state forRegion:
(CLRegion *)region
{

    if (state == CLRegionStateInside)
    {
         // Perform tasks for entering the region
    }
    else
    {
         // Perform tasks when exiting the region
    }

}
```

Monitoring for iBeacons is the fundamental capability that allows for interaction between an application and the physical environment. Core Location makes triggering actions simple, with a well-designed framework for writing a wide variety of custom triggers that could be required by an application.>

Monitoring Limitations

The Core Location framework does not always provide immediate updates of region boundary-crossing events. When devices are at the edge of an iBeacon's range, reception is marginal. Rather than send rapid-fire events as a receiver is at the ragged edge, the Core Location framework will damp out excessive numbers of triggered events.

iOS can monitor for up to 20 UUIDs in the background, so complex applications might need to modify the monitor list to add more specific monitoring when a device is in proximity to the UUID. For example, an application can monitor for a UUID and, only once that UUID is detected, add monitoring for particular major and minor numbers.

iOS 7.1 brought a major change in monitoring. Prior to iOS 7.1, monitoring could be performed by an app only when it was running. In the 7.1 update, monitoring can now be used to launch an application that is not running.

Monitoring requires that a user enable location services on iOS and, if necessary, also enable location information for the specific application.

Regions are tracked by the operating system, not the application. Even when applications are not running, iOS may deliver region-crossing events. If an application has stopped or is running in the background, the OS can relaunch or awaken the app to handle the event.

Ranging

The second major type of interaction with an iBeacon is *ranging*, which is used to determine the distance between a device and a particular iBeacon. Like monitoring, ranging is conceptually simple with Core Location:

1. As with monitoring, create a `CLBeaconRegion` object.
2. Create a `CLLocationManager`, but rather than monitoring for region boundaries, ask it to perform ranging. Range updates are supplied every second.

The Core Location framework summarizes the range into a general description of *immediate* (within a few centimeters), *intermediate* (a few meters), and *far* (the iBeacon's transmissions are received, but it is more than a few meters away).

The following code creates a `CLBeaconRegion` object and starts ranging:

```
NSUUID *myUUID =
   [[NSUUID alloc]
   initWithUUIDString:@"24F7B8B3-D124-4B7B-A180-CBA317CC1BB6"];
CLBeaconMajorValue myMajor = 42;
CLBeaconMinorValue myMinor = 105;

// This region matches on one specific beacon in the UUID
CLBeaconRegion *matchThree = [[CLBeaconRegion alloc]
               initWithProximityUUID:myUUID
               major:myMajor
               minor:myMinor
               identifier:@"matchThree" ];

CLLocationManager *myLocationManager =
                  [[CLLocationManager alloc] init];

// Change to monitor for a specific iBeacon within that UUID
[myLocationManager startRangingForBeaconsInRegion:matchThree];
```

To access ranging data, the framework will trigger the `didRangeBeacons` method. An array of iBeacons that have ranging data will be passed to the delegated method, along with the region in which they were detected. It is also possible to use the received signal strength indicator (RSSI) to estimate a range in meters, which is a property of the `CLBeacon` object in the array.

The following simple method loops over each iBeacon's data and prints out a message based on how far away it is:

```
- (void)locationManager:(CLLocationManager *)manager
          didRangeBeacons:(NSArray *)beacons
          inRegion(CLRegion *)region
{

   UILocalNotification *message =
     [[UILocalNotification alloc] init];
   message.soundName = @"Default";

   for (CLBeacon *eachBeacon in beacons)
       {
       if (eachBeacon.proxmity == CLProximityImmediate) {
          message.alertBody = @"On top of the beacon!";
       } else if (eachBeacon.proxmity == CLProximityNear)
       {
          message.alertBody = @"Near the beacon!";
       } else // proximity is "far" if you get here!
       {
          message.alertBody = @"This beacon is in range!";
       }
   }
```

```
        [[UIApplication sharedApplication]
          scheduleLocalNotification:message];

    }
```

iOS places two important limitations on ranging. Most importantly, ranging must be done in the foreground. Applications running in the background cannot perform ranging.[4] Furthermore, the Core Location framework cannot perform triggering based on ranging, so it is not possible to take an action when a device is a certain distance away from a beacon.

iBeacon Triggers for Passbook

iBeacons can trigger actions from applications running on the device, but not every organization has the time, expertise, desire, and funding to develop an application. To provide basic interactions, the Passbook application has the ability to create passes that interact with iBeacons.

When the device is near an iBeacon, a pass with the iBeacon's identification information can be displayed on the lock screen, and the pass can be updated based on the visit. The first step in updating a pass based on proximity is to activate the pass when it enters the range of an iBeacon.

One method, shown in Figure 5-2, is embedding an iBeacon's identifying information into the pass. The right side of the figure illustrates the location parameters that will be embedded into the pass. When the pass is within range of the UUID, major number, and minor number, the user will receive an alert. With additional support from a network service, the pass can be updated based on any information available to the computer systems.

4 Ranging operations require much more activity in the Bluetooth hardware and draw significant power, because the Bluetooth interface is much more active when ranging.

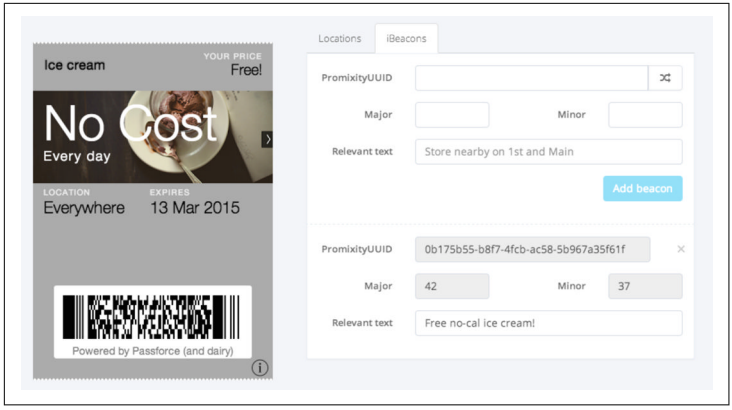

Figure 5-2. Linking a Passbook pass to an iBeacon

Regardless of whether your choice is to write a full application, or to go with a simpler pass-based interaction, correctly planning out the support network is an important consideration.

CHAPTER 6

Building iBeacon Networks

iBeacons are simpler than other wireless networks, and as a result, the planning process can be much more lightweight than what you might be expecting if your only experience working with wireless is with higher-bandwidth networks such as Wi-Fi.

Building an application and putting together a supporting iBeacon network are two sides of the same coin. Whether the application is designed to help shoppers find items, travelers navigate an airport or hotel, or conference attendees register and find sessions of interest, iBeacons must be deployed in physical space to support that application.

Planning and Project Objectives

As with any other project, the key to success is to have a plan with clear objectives. In the case of an iBeacon project, the main purpose is to build support for an application running on a mobile device, and any physical network design aspects must take their cue from application requirements. Objectives set by the application development team will inform many of the design decisions made for the initial deployment as well as ongoing maintenence.

Identifying the Objective

Begin planning with identifying the objective of the iBeacon project. Objectives can be varied, depending on the purpose of the application under development. Some common motivations to strive for

are increasing customer engagement and making other products or services easier to use.

In the broadest possible sense, *engagement* is a measure of how much users are interacting with your application. When applications are newly released, organizations will often use engagement, such as the amount of time spent actively using an application, as a key metric in success. For an iBeacon project, the goal is often to increase the amount of time users spend with an application by adding interactivity to it. A retailer might use iBeacons to drive in-store shoppers to their website to look through the full catalog, and they might track the success of an application based on how many shoppers use the website based on an in-store trigger.

Ease of use is a broad category, and there are many ways for iBeacons (or in fact any proximity technology) to bridge the gap between the physical and virtual worlds. As many more small sensors and controls are introduced to homes, the setup process is increasingly daunting for many people. Because they operate at short range, iBeacons are useful as a trigger to begin configuring a nearby device, perhaps by triggering a higher-bandwidth connection such as Wi-Fi.

Proximity enables applications to automate business processes, decreasing the amount of time spent waiting. Whether an application can submit an online restaurant order automatically on entering a restaurant, print a conference badge automatically when the user steps up to the registration desk, or trigger the completion of a transaction, proximity pulls application users into your processes and makes them easier and faster to complete.

An underdeveloped area of iBeacon application is peer-to-peer applications. iBeacons themselves cannot broker communications, but they enable applications to make direct connections through an external rendezvous point. A few applications register the iBeacon identities that they are near and then use an IP-based service to make direct connections to other devices nearby. Although a few applications have started to take advantage of this capability, it remains largely unexploited.

Selecting Numerical Identifiers

With a clear idea of the purpose of the application, the next step is to design the interaction between iBeacons and the application. An

application responds to identifying numbers in iBeacon transmissions, so as part of designing and building the application, you will need to select those numbers and figure out how to assign them:

UUID
: The UUID is the easiest number to come up with because it should be unique to the application. An application that supports multiple brand names might use different UUIDs, but generally speaking, the UUID is consistent across the world for the entire application. You can easily get one by using the `uuidgen` tool on the command-line interface of your Mac.

Major number
: Within the context of an application, the major number needs to identify broad groups of proximity areas that make sense logically. Within a chain of retail stores, the major number is typically tied to a store and the 16-bit field allows for 65,000 possibilities. In a more fragmented market, such as selling services to individual small businesses, the major number could potentially be a subscriber to services, but if the service is successful and signs up more than 65,000 customers, the numbering scheme might need to be redesigned.[1]

Minor number
: The minor number is a further subdivision of the grouping defined by the major number. Major numbers typically refer to a geographic location, and the minor number would be points of interest within one location. Running out of minor numbers seems unlikely in most circumstances.

Do not depend on iBeacons themselves to provide any security. iBeacon broadcasts are readily received by devices. The Radius Networks tools from Chapter 3, and many similar tools, will report on observed iBeacons, so you should expect sophisticated users to decode the iBeacon triggers in your application.

1 While writing this book, I looked at a popular restaurant reservation service and found that they had 31,000 restaurants using their service, which means that the company might not want to use the major number for an individual restaurant. As far as I can tell, it is currently safe to use the major number for a chain; the largest restaurant chain I found has a little over 42,000 restaurants.

In fact, Radius has begun to record iBeacon locations and has made them available on a map at WikiBeacon. Figure 6-1 shows an artistic iBeacon arrangement of a beaker, overlaid on a Google Map.

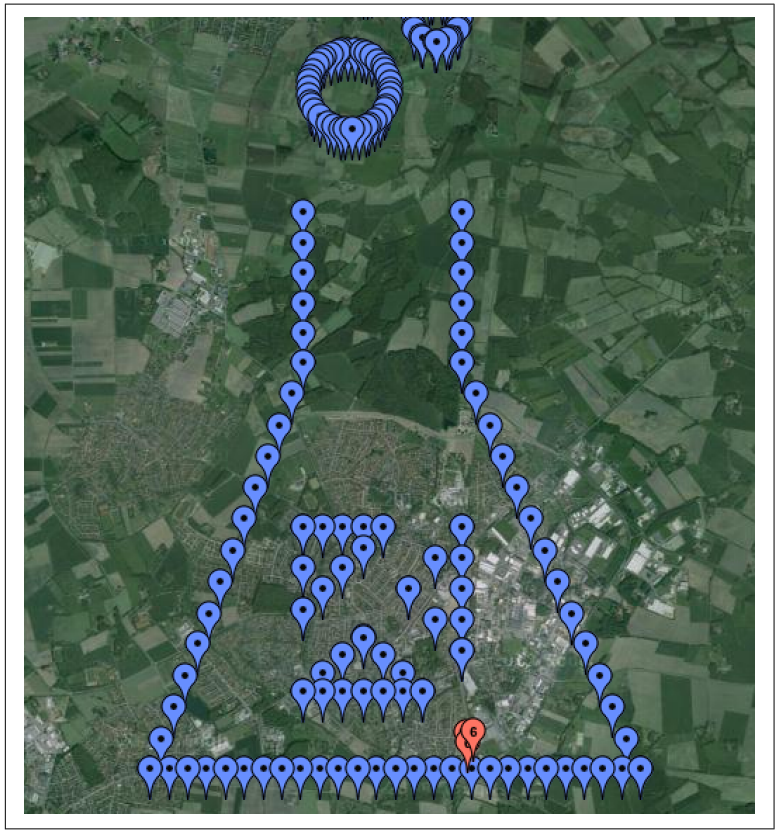

Figure 6-1. iBeacon beaker

Beacon Location Selection

To support the application, iBeacons need to be placed into the physical space that the application interacts with. The number of iBeacons depends on the objectives of the application. Relatively simple applications might require only a few iBeacons, but complex applications might require a large number of proximity points.

Both fixed and mobile locations can be used. The former is useful for helping application users navigate to locations within the space, while the latter option is used to make an application respond to

proximity. Using mobile iBeacons lets the application automatically adapt to changes in the physical space. If, for example, an application uses iBeacons to monitor queue length (say, of an airport security line), the beacons can change locations without affecting the function of the application. iBeacons still trigger an application to log queue entry and exit, even if the queue changes its location within the building.

Choosing locations depends on the number of proximity points that an application requires. For a basic "welcome to our building" application, a few iBeacons at high power can readily blanket an area with beacon transmissions. Look for locations up high and with an unobstructed line-of-sight view to the application users. More sophisticated applications that help with wayfinding or trigger actions based on being within a few feet of something require many more iBeacons working at lower power.

The good news is that selecting locations for iBeacons is much easier than for data-oriented networks such as Wi-Fi. Bluetooth signals require significantly lower signal-to-noise ratios, employ frequency-hopping to avoid interference, and operate at much lower power. Many iBeacons can run for long periods of time on battery power, which is ideal for an iBeacon attached to a retail display.

iBeacons can even be designed to move, depending on the purpose of the application. If an application is designed to help interact with a museum artifact, the iBeacon will move with its display so that users can still find relevant information no matter how its location changes.

Beacon Configuration

iBeacons do not require much in the way of configuration because of the simplicity of the protocol. Essentially, an iBeacon needs to be configured with its identifying numerical tuple (UUID, major number, and minor number), along with a power setting and a calibration constant to turn received signal power into an accurate proximity measurement.[2]

2 Many iBeacon devices also allow configuration of the advertisement interval, but Apple's iBeacon specification fixes it at 10 advertisements per second.

Setting the power on an iBeacon depends on the desired effect. High power provides a large coverage area. If an application is designed to automatically pay for a waiting order, it is likely that the iBeacon that triggers that process should use high power to begin the payment transaction as soon as possible. However, if the purpose of the application is to guide a user through a museum, the power setting likely should be smaller to reduce the number of exhibits an application might display interpretive text for.

High- and low-power iBeacons can be mixed within a deployment as well. High-powered transmitters can be used to wake up an application while the user is still walking up to the entry point, so that the application then begins actively searching for triggers based on specified major and minor numbers.

At this point, configuring large numbers of iBeacons is a time-consuming experience. With few exceptions at the time of this writing, the available tools are designed for configuring individual iBeacons, so there is no good way to make changes to multiple devices at once. Each iBeacon also needs to be calibrated. To build iBeacon networks at a large scale, management tools need to adopt a network-centric view and the capability to configure multiple devices at once, preferably with the ability to assign major and minor numbers based on location.

Monitoring

Once an iBeacon network is set up, there is the need to run the network on an ongoing basis. iBeacons do not present a significant ongoing monitoring burden. The following list details the major items that need to be resolved:

iBeacon disappearance
: Applications are designed to look for a set of iBeacons and take actions based on them. Once installed, iBeacons should continue to function. iBeacons might disappear when they are removed from service, such as when an exhibit is removed or a promotion ends. Premature disappearance is likely because a battery-powered iBeacon has stopped operating, and action is required to restore the iBeacon to its transmitting state.

iBeacon appearance
: The flip side of iBeacons going away is that iBeacons that appear should also be checked. Either they are part of a new display

and should be tagged as such, or they might represent rogue beacons that should be found and deactivated.

iBeacon health
: Battery-powered iBeacons might give warnings of impending failure as their battery capacity decreases. If an application requires large numbers of battery-powered iBeacons, some sort of mass-monitoring tool to check on battery health is a practical requirement.

Project Checklist

iBeacon projects are exciting because they offer the opportunity to make an organization's technology infrastructure respond more smoothly to users and make organizational processes more efficient. Much of an iBeacon project is based around developing an application that responds to its surroundings.

Being successful with iBeacons requires attention to a few key details:

Clearly identify project objectives
: With iBeacons, the "big idea" is to use proximity to enhance an application user's experience.

Design the iBeacon network
: Choose the right structure of the numerical identifiers, and place iBeacons in the right number of proximity points to support your application.

Configure and monitor the iBeacons
: At the time this book went to press, large-scale monitoring and configuration were unsolved problems. It seems likely that moving from managing individual devices to managing groups of devices is a necessary development to support large iBeacon projects.

With the application developed and the supporting cast of iBeacons in the physical space, you are ready to let users at the application!

Index

A

activating iBeacons, 22-29
  on iOS devices, 22-24
advertisement interval, 15, 57
  battery life and, 20
  in iBeacon specification, 20
advertising events, 14
advertising packets, 16-18
  defined, 13
  header, 16
  nonconnectable undirected, 14
Allan, Alasdair, 38
appearing iBeacons, 58
Apple Company ID, 17
application development, 31-40
  limitations, 32
  monitoring functions, 34
  monitoring lists, changing, 34
  privacy, 39
  ranging functions, 35-37
  security and, 38
  web services interactions, 37
applications, 5-9
  indoor direction finding, 6
  proximity-triggered actions, 7
  queue management, 8

B

battery life, 20
  advertisement interval and, 20
  beacon disappearance and, 58
  monitoring, 59
  of Estimote beacons, 20
  range finding and, 36
beacon track, 39
Beacon type field (iBeacon payload), 17
Bluetooth interface (Raspberry Pi), 26
Bluetooth Low Energy (BLE) beacons, 9
  advertising interval of, 15
  advertising packet contents, 16-18
  configuring iBeacon Protocol for, 13-16
  specifications, finding, 16
  Universal Unique Identifier (UUID), 14
  WiFi versus, 57
Bluetooth SIG, 17
  specifications, finding, 16
BlueZ stack, 27
Bring Home the Beacon Hackathon, 31
broadcaster, Bluetooth, 13
Broadcom BCM20702A0 chipset, 26

C

calibration and ranging, 36
California Department of Motor Vehicles, 9
car locater application, 6
CLBeacon objects, 49
CLBeaconRegion objects, 44, 48
CLLocationManager objects, 45

Company ID field (advertising header), 17
list of, 17
Core Location framework, 41-51
actions in, 45
documentation, 41
event handling in, 45-47
limitations on, 47
monitoring with, 43-48
ranging and, 48-50
cross-references security measure, 39
Cupid's Kiss (Canova), 4

D

dedicated hardware, 19-21
Department of Motor Vehicles (California), 9
device-to-device communication, 32
didDetermineState method, 46
didEnterRegion method , 46
didExitRegion method, 46
disappearing iBeacons, 58
distance, calculating, 18
dynamic data storage for applications, 37

E

engagement, 54
Estimote beacon hardware, 19
event handling
in applications, 37-38
in Core Location, 45-47

F

fixed locations for iBeacons, 56
Flags field (advertising header), 16

G

Gelo beacon hardware, 20
general-purpose hardware, 21
geofencing, 38
geographical information, accessing, 33
Global Positioning Systems (GPS)
accuracy of, 5
in parking garages, 6

H

hardware, iBeacons, 19-21
dedicated, 19-21
general-purpose, 21
hciconfig, 28
HDMI cables for Raspberry Pi, 27
Hong Kong, 6
hysteresis, implementing, 35

I

iBeacon Protocol, 13-18
advertising interval, 15
advertising packet, 16-18
Apple and, 2
configuring, 13-16
major number, 14
minor number, 14
payload, 17
Universal Unique Identifier (UUID), 14
iBeacons, 1-11
activating, 22-29
advertisement interval for, 20
application development for, 31-40
applications using, 5-9
basic functions for, 33
BLE beacons as, 9
building networks of, 53-59
Company ID for, 17
configuring, 57
fixed locations for, 56
indoor direction finding with, 6
interactivity of, 5
limitations on, 32
location versus proximity, 3-5
locations, art created with, 56
mobile locations for, 56
monitoring functions, 34, 43-48
monitoring limitations, 47
monitoring rules, 35
monitoring with Core Location, 43-48
on iOS, 41-43
privacy, 39
receiving data on, 32
security of, 38
selecting locations for, 56

setting up, 19-29
usage, 2
iMac, 21
indoor direction finding, 6
interactivity of iBeacons, 5
IOGEAR GBU521 Bluetooth interface, 26
iOS 7, 21
iOS devices as iBeacons
activating, 22-24
Core Location framework, 41-51
laptops/desktops, 21
iOS Location Services Control Panel, 42
iPad 3, 21
iPad mini, 21
iPhone 4s, 21
iPod touch, 21
isMonitoringAvailableForClass method, 45

K

"The Knowledge" test, 1
Kontakt beacon hardware, 20
Kytelabs BLEduino, 21

L

large-scale iBeacon networks, 58
Length field (advertising header), 17
Linux on Raspberry Pi, 21
Raspbian distribution, 27
local storage (web service interactions), 37
Locate for iBeacon app (Radius Networks), 22
transmitter screen elements, 23
location
fixed/mobile, 56
managers, 45-47
proximity versus, 3-5
selecting, 56
services, 37
user permission for use, 42

M

MacBeacon program, 21
MacBook Air, 21
MacBook Pro, 21
major number, 14
as iBeacon payload, 18
matching, 34, 43
selecting, 55
map replacement in large buildings, 6
measured power field (iBeacon payload), 18
ranging calibration and, 36
micro-locations, 4
minor number, 14
as iBeacon payload, 18
matching, 34, 44
selecting, 55
Mistry, Sandeep, 38
mobile advertisement application, 7
mobile devices
ranging power requirements, 33, 50
sending messages to, 33
mobile locations for iBeacons, 56
monitoring, 33
alerts, 35
defined, 33
rules, 35
monitoring functions
limitations on, 47
with Core Location, 43-48
multipath interference, 36
Musée de la Musique in Paris, 7
museum guides application, 7

N

nearby clients, reporting on, 32
networks, building, 53-59
configuring beacons, 57
large-scale, 58
location selection, 56
monitoring, 58
numerical identifiers, selecting, 54
objectives, identifying, 53
nonconnectable undirected advertising packets, 14
Nordic Semiconductor, 21
numerical identifiers, selecting, 54

O

outdoor iBeacons, 20

P

Passbook, iBeacon triggers for, 50
patient information integration applications, 8
PayPal, 9
Penn Station (New York City), 6
pirate beacon, 38
power setting, 24
  selecting, 58
power sources, 20
  for low transmit power devices, 10
  for Raspberry Pi, 26
  transmission power and, 10
privacy, 39
proximity
  beacons, 2
  estimation, 14
  location versus, 3-5
  RSSI and, 14
  -triggered actions, 7
Proximity Kit (Radius Networks), 39
proximity UUID field (iBeacon payload), 18

Q

queue management applications, 8

R

RadBeacon
  app, 24
  configuring, 24-26
  hardware, 20
Radius Networks, 22
  iBeacon location recording, 56
  Proximity Kit, 39
ranging
  accuracy and calibration, 36
  code samples for, 48
  Core Location and, 48-50
  defined, 33
  function, 35-37
  measured power field (iBeacon payload) and, 18
  multipath interference and, 36
  power requirements for, 36
Raspberry Pi
  as iBeacon, 21, 26-29
  hardware requirements for, 26
  installation, 27-29
  power source for, 26
  setup, 27-29
Raspbian distribution of Linux, 27
received signal strength indication (RSSI), 14
  estimating range with, 49
RedBearLab BLE Mini, 21
regions
  boundary-crossing events, 47
  monitoring for, 34
requestStateForRegion method, 46
retail applications
  mobile advertisement, 7
  store enhancement, 7
  transaction completion, 9
Robinson, Frank, 1
rogue beacons, 58
RSSI (received signal strength indication), 14
  estimating range with, 49

S

SD card for Raspberry Pi, 26
security, 55
sensor fusion, 1
Shinjuku Station (Tokyo, Japan), 6
Square, 9
static data storage for applications, 37
subways, 6

T

table pager application, 9
ticket validation applications, 8
timestamping security measure, 39
Tokyo
  Shinjuku Station, 6
  subway system, 6
transaction completion application, 9
transit assistance application, 6
transmitter screen for Locate for iBeacon app, 23
treasure hunt applications, 8

Type field (advertising header), 17

U

Universal Unique Identifier (UUID), 14
creating, 15
Estimote beacons and, 20
matching, 34, 43
selecting, 55

V

Venus de Milo, 4

W

web services interactions, 37
The Wedding Feast at Cana (Veronese), 36
WiFi versus Bluetooth Low Energy, 57

About the Author

Matthew S. Gast is the director of product management at Aerohive Networks, responsible for the software that powers Aerohive's networking devices. He has been active within the Wi-Fi community, serving as the chair of both security task groups at the Wi-Fi Alliance, where he leads efforts to extend the Wi-Fi Protected Access (WPA) certification to incorporate newly developed security technologies and drive adoption of the strongest forms of security by network administrators. He also led the Wi-Fi Alliance's Wireless Network Management marketing task group's investigation of certification requirements for new power-saving technologies. Matthew is also the past chair of the task group that produced the 802.11-2012 revision.

Colophon

The animal on the cover of *Building Applications with iBeacon* is a vampire bat. The three species of bat to which this generic name refers—the common vampire bat (*Desmodus rotundus*), the hairy-legged vampire bat (*Diphylla ecaudata*), and the white-winged vampire bat (*Diaemus youngi*)—possess a highly specialized profile that distinguishes them markedly from other bat species. Chief among their distinctive qualities is the dietary requirement for blood, which vampire bats obtain from live mammals and birds while hunting in the darkest nighttime hours. Like pit vipers, they are neurologically equipped to locate blood near the surface of their victims' skin.

Though solitary as hunters, vampire bats demonstrate strong familial ties to the other bats in their colonies, which are found throughout the Central and South American tropics. Bats of the same colony have been known to practice food-sharing and even to adopt young in the colony that have lost a mother.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover image is from *The Riverside Natural History*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.