SILICON LABS

Developing Beacons with Bluetooth® Low Energy (BLE) Technology

Developing Beacons with Bluetooth® Low Energy (BLE) Technology

Table of Contents

Executive Summary

Bluetooth® beacons are taking off. They enable “proximity-aware applications” for customers, businesses, and industrial environments.

- End customers benefit through instant coupons and tailored offerings based on where they are.
- Businesses benefit through improved visibility to customer buying habits and increased loyalty.
- Industrial companies benefit through improved asset monitoring and utilization.

The possibilities are endless, and beacons are set to transform our world. But before they do…you should know that implementing them can be challenging. Putting beacons on a product, pushing their data into the cloud and then using it create value all represent new development frontiers for many of us.

It’s not straightforward. Did you know that Bluetooth beacons are not, in fact, a Bluetooth standard? Bluetooth beacons are pseudo-standards running on Bluetooth’s low energy technology (previously known as Bluetooth Low Energy, BLE, or Bluetooth Smart), but they use proprietary beacon code beyond that.

This paper covers a lot of territory.

- We examine beacon applications to help you brainstorm some of your own.
- We provide a short history of Bluetooth and its derivatives, including Bluetooth low energy and beacons.
- We cover the leading beacon pseudo-standards at a high level, and in detail in the Appendix.
- We provide references to field-hardened example code and tools to develop and deploy it.
- And we provide information on end-to-end solutions to get you started.

To begin, once a beacon developer is ready to start, they need to focus on several key items.

1. Select a widely-adopted, proven Bluetooth stack. Because beacons are new, there is room for FUD from new entrants. Buyer beware. Use a company that understands low-energy Bluetooth and has the market adoption to prove it.
2. Select a company with proven software development tools. Software development is more than just an editor and debugger. It also needs to include mature example code for common use cases, scripting engines for common commands, and thoughtful APIs for ultimate migration from one standard revision to another. These tools will all work together to speed development.
3. Select a company with good customer support. This goes hand-in-hand with a market-proven stack and software tools; if the selected company is market-proven, then they have support forums with hundreds of questions and answers on hardware and software, an adequate user base and support staff to answer new ones, a broad portfolio of product variants for different applications, and so on.

Good luck! Read on…

Introduction

Bluetooth® beacons will have a transformative impact on the way we interact with the physical world. They enable proximity-based contextual awareness using technology that most of the world's population now carry in their pocket – a smartphone with applications.

Using wireless technology for proximity detection is not new, but with the introduction of Bluetooth's low energy features[1] in 2009, beacons are now being deployed on a wide-reaching scale.

This article explains how beacons work, some of the ways they can be used, and design considerations for beacon product development.

What is a Beacon and How is it Used?

In general terms, a beacon is a small, battery-powered, wireless device that uses Bluetooth low energy technology (Bluetooth Smart) to advertise its presence and services. It does this by repeatedly broadcasting or advertising a beacon identifier to compatible smartphones or tablets within its proximity. The smartphone or tablet can then use the beacon's information to determine its location and services, and act accordingly.

Beacons enable proximity-based customized experiences for users.
Can't find the Tylenol? Need a quick sandwich on the way to your flight?

Beacons can help.

Beacons are generally used for proximity-aware applications. By monitoring beacons, a device can detect when it has entered or exited a particular area, and then use that information to create interactive experiences based on what's nearby. The group proxbook.com provides a list of such proximity-aware applications.

There is no official Bluetooth Special Interest Group (SIG) beacon standard. Instead, there are various beacon pseudo-standards from company consortiums or large operating system providers. Each pseudo-standard takes advantage of some of the Bluetooth low energy technology's native facilities and the widespread availability of Bluetooth itself. The more prominent pseudo-standards are Apple's iBeacon™, Google's open source Eddystone™, and Radius Networks' AltBeacon.

Platform	Bluetooth low energy technology support (BLE)	Native* Beacon Support
Apple iOS	iOS 7.0 (2013)	iBeacon
Apple Mac	OS X	iBeacon
Google Android	4.3	Eddystone
* Note that this table shows only *native-OS* support. Each major OS supports other beacon types as well with relevant applications.		

OS support for Bluetooth low energy technology and beaconing pseudo-standards

[1] The Bluetooth Special Interest Group (SIG) recently rebranded to remove sub-brands Bluetooth SmartReady and Bluetooth Smart to reflect only "Bluetooth" with common language to describe any relevant sub-features. For example, "Bluetooth Smart" is now referred to as "Bluetooth low energy technology." See Bluetooth.org for more information.

Two Beacon Usage Models

Beacons are typically used in one of two scenarios. The first and most common is for a beacon to be placed either in a fixed location or on a movable object, then relying on a smartphone to correlate beacon proximity to a desired behavior such as opening an app or offering contextually-relevant content.

The second uses a fixed wireless node to monitor beacons on objects that pass by or through its monitoring area. It then can report back to another application using a wired or wide-area network. This model might apply to asset tracking for expensive tools and equipment, livestock, or even people (wearing a bracelet tag for example).

Example fixed location beacon

Proximity-Aware Example Applications

The two usage scenarios above rely on proximity awareness. In the first scenario, a user smartphone comes into proximity with a beacon. In the second, beacons come into proximity with a beacon-monitoring node. Both models are finding applications in retail and commercial businesses.

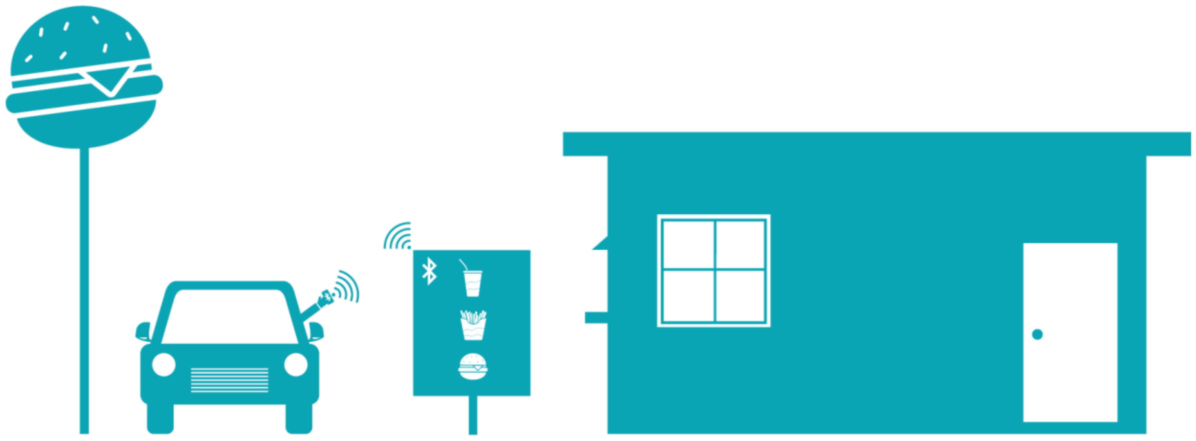
The most established applications focus on retail shopping. Beacons distributed throughout a store allow loyalty apps to offer personalized experiences to its customers. The applications serve tailored messages and coupons and track the customer’s reactions for additional customization.

A smartphone can detect when it enters or exits a particular setting by monitoring beacons, then use that information to create interactive experiences based on nearby products and services.

Other applications include point-of-sale systems such as vending machines. For example when a customer approaches a beacon-enabled vending machine, the customer’s smartphone activates a payment app or website to suggest favorite items or a menu of available options.

Beacons also enable more secure payments. In the vending machine example, the customer’s smartphone app can recognize the beacon-tagged machine to allow the customer to pay for their selection using the cellular network. The cellular wireless connection is more secure than a local one, and they never have to remove their wallet!

This same use case is emerging at fast-food drive-throughs. When the driver approaches the beacon-enabled drive-through equipment, their smartphone detects the beacon and activates customized offers. The same application can also optimize inventory management based on customer usage.



Beacon-enabled drive through

Commercial beaconing applications are also gaining momentum. As mentioned above, beacons can track and help manage important assets like expensive power tools. A beacon-enabled tool allows it to “check in” to a monitoring node to determine when it is in a tool bin, on the shop floor, or not in range. The same beacon application can monitor and report tool status such as charge level, operating time, and performance. This has obvious implications to the lifetime and security of the tool as well as its optimized utilization.

Additional beaconing applications provide indoor navigation and relevant localized content in large buildings such as hospitals, shopping malls, or museums.

Imagine viewing the Mona Lisa at The Louvre Museum in Paris, only this time she has a beacon. Your smartphone would detect the beacon and present you with The Mona Lisa Wikipedia page, a biography of Leonardo da Vinci, and a history of the famous Peruggia theft of 1911. Finally we know why she is smiling!

Now we know why Mona Lisa is smiling…she has her own Bluetooth beacon!

Beacon Proximity Accuracy

Obviously, proximity-aware applications rely upon knowing which beacons are nearby. But a beacon's RF range can vary from under a meter, to just a few meters, to over 500 meters depending on its transmit power[2]. This makes it impractical to determine proximity based solely on receiving a beacon identifier packet.

For this reason, beacon proximity relies on a comparison of a Received Signal Strength Indicator (RSSI) to a beacon's transmit (Tx) power to approximate the distance to the beacon. Beacons provide a calibrated transmit (Tx) power figure in their advertising packets. Coupling this information with the RSSI enables a receiving device to approximate how far away the beacon is. Unfortunately, this calculation cannot be very accurate since RF signals fade unpredictably according to real-world environmental factors like walls, weather, and people. Luckily, approximate distance is generally good enough for proximity applications since the typical intent is just to determine whether the beacon is close by or far away (e.g., 1 meter, 10 meters, or 100 meters).

[2] Bluetooth low energy transmit power was limited to 10 mW until Addendum 5 was released in December 2015, increasing the maximum allowable Tx power to 100 mW, and providing up to 2.5x more range.

Future versions of the Bluetooth specification will likely incorporate Angle-of-Arrival (AoA) and Angle-of-Departure (AoD) features which allow a multi-antenna Bluetooth device to accurately determine the spatial location of another Bluetooth device[3]. These features will enhance Bluetooth's usefulness in applications requiring high-accuracy location detection, potentially giving position accuracy to within tens of centimeters. This level of precision would enhance the effectiveness of beacons when used in asset management applications.

A Short History and Technical Overview of Bluetooth Low Energy Technology

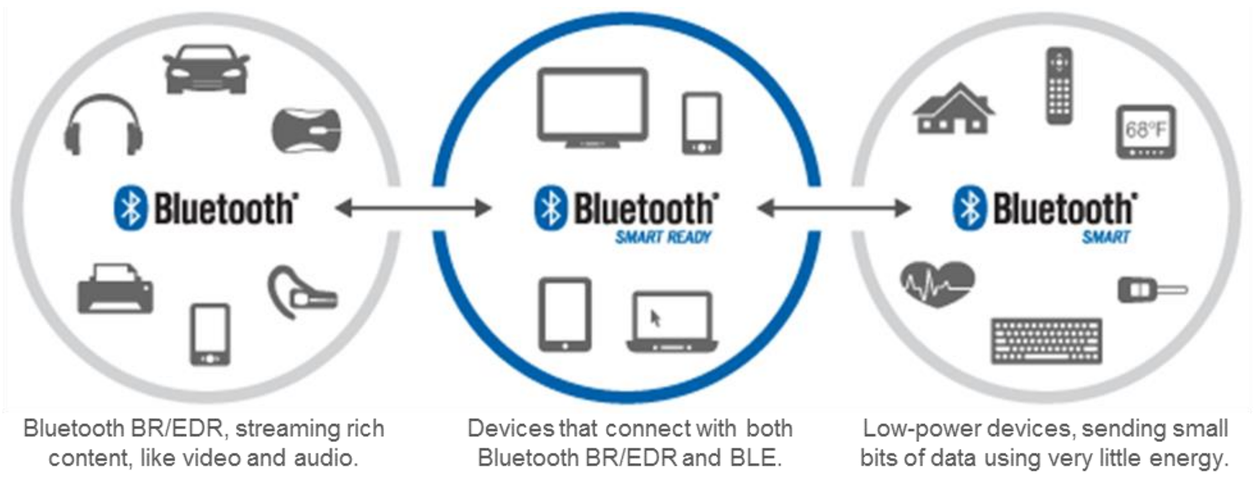
The Bluetooth SIG announced Bluetooth low energy features with version 4.0 of the Bluetooth specification in 2009. It was later re-branded as Bluetooth Smart, and has just been rebranded again in 2016 (see footnote 1 above). Consequently the terms Bluetooth Smart, Bluetooth Low Energy, BLE, and now Bluetooth low energy technology are often used interchangeably.

Bluetooth low energy technology is a radical departure from what is known as Bluetooth Basic Rate / Enhanced Data Rate (Bluetooth BR/EDR) or Classic Bluetooth, introduced in the late 1990s and used today in handsets, speakers, earphones, car kits, etc. The SIG's goal with Bluetooth low energy technology was to define a new version of Bluetooth which could operate for years on a coin cell battery and was better suited for sending small bits of data on an infrequent basis.

Bluetooth BR/EDR is still the protocol of choice for voice or streaming music, but Bluetooth low energy technology is better suited to wireless sensor and control applications. Bluetooth low energy technology also reduced data latency to only 10% that of Bluetooth BR/EDR and introduced the ability for Bluetooth to broadcast data.

Like Bluetooth BR/EDR, Bluetooth low energy technology utilizes the 2.4 GHz ISM band and a frequency hopping technique to spread its RF energy between multiple channels. But in a departure from Bluetooth BR/EDR, it uses 40 2MHz-wide channels instead of 79 1MHz-wide channels. Consequently the two versions are fundamentally incompatible over the air. Devices that can support both Bluetooth BR/EDR and Bluetooth low energy technology have been called "Bluetooth Smart Ready" up until March of 2016. Most handsets, tablets, and so on fall into this category.

[3] AoA and AoD features will tentatively be supported in Bluetooth v5.0


Bluetooth
Bluetooth SMART READY
Bluetooth SMART
68°F
Bluetooth BR/EDR, streaming rich content, like video and audio.
Devices that connect with both Bluetooth BR/EDR and BLE.
Low-power devices, sending small bits of data using very little energy.

Bluetooth BR/EDR, Bluetooth Smart Ready, and Bluetooth Smart (BLE) (See footnote 1 above)

Three of Bluetooth low energy technology's 40 channels (37, 38, and 39) are reserved for broadcasting "advertising packets" that contain information about the broadcasting node's capabilities. These advertising packets are strategically located on frequencies between the three 2.4 GHz Wi-Fi channels to avoid interference from Wi-Fi.

Bluetooth Generic Attribute (GATT) Profile

The SIG also greatly simplified the application profiles for Bluetooth low energy technology, which now uses the GATT (Generic Attribute) profile, a structured list that defines the services, characteristics and attributes of a given application. GATT profiles can use either SIG-defined services or custom services defined by the OEM. Each GATT profile's service is distinguished by a Universally Unique Identifier (UUID) which is either 16 bits long for SIG-adopted services or 128 bits long for custom services defined by the developer.

A beacon can include multiple services. When a service needs to be advertised, the service UUID is broadcast in the device's advertising packet. Subsequently, when a Bluetooth scanner receives an advertising packet, the UUID is registered by the operating system to a specific application, which takes follow-on actions.

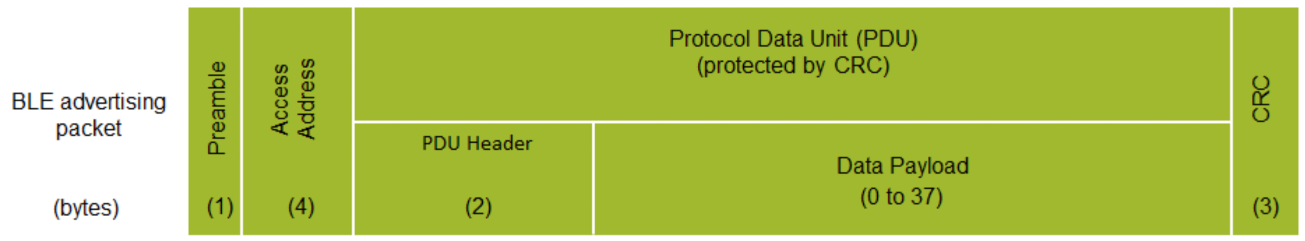
The Bluetooth Low Energy Technology Advertising Packet

Beacons work by taking advantage of Bluetooth's ability to broadcast packets with a small amount of customizable embedded data on its three advertising channels. A Bluetooth low energy scanner routinely scans for advertising packets and then decodes them to ascertain the content and take appropriate action.

Bluetooth advertiser nodes use advertising packets in channels 37, 38, and 39 to advertise their supported GATT profiles.

Scanners receive the advertising packets, decode them, and take action.

Both advertising packets and data packets use the same format (see figure below). Beacons follow the standard advertising packet format, but then format the embedded data payload to follow a pre-defined structure for the beacon pseudo-standard it uses. This allows the OS to treat a beacon's advertising packet differently from other Bluetooth advertising packets.



Bluetooth low energy packet format is the same between data packets and advertising packets

Bluetooth low energy devices transmit advertising packets on a selectable interval as short as 20 milliseconds or as long as several minutes. The same packet is sent on all three of the advertising channels every time the device advertises, making it more likely that a scanner will pick it up.

Within the advertising packet, the data payload is structured as one or more [length, type, data] triplets.

- The length field defines the combined size of the subsequent type and data fields;
- Followed by the type field which designates whether the data is a name, a service UUID, a Universal Resource Identifier (URI), or one of many other defined data types; and
- The packet data itself.

Beacons take the structure a step further, defining a sub-structure inside the data field as shown in the following sections on the various pseudo-standards.

Transmitting a single beacon packet can take up to 376µs[4], but could be shorter depending on which beacon standard is followed. How frequently a beacon advertises is a trade-off between power consumption and the tolerable application latency.

In a more conventional*, <u>non-beaconing</u>* Bluetooth low energy application, advertising packets provide identifying information about the advertiser's services and are then followed by a period of time in which the advertiser actively listens for a connection requests from scanners which want access to those services.

However, beaconing Bluetooth applications will typically use non-connectable advertising, essentially fire-and-forget, providing all of their useful information in the advertising packet itself. Since they generally don't listen for connection requests, the radio can be shut off immediately after advertising, further extending battery life.

[4] This assumes a fully utilized advertising data payload of 39 bytes.

It’s important to note that some beaconing applications include other GATT services in addition to the beacon, and those services may interleave connectable advertising packets with the non-connectable beacon packets. In this case the beacon performs a more typical advertising cycle with transmit/listen operations.

Beaconing Pseudo-Standards (iBeacon, Eddystone, AltBeacon)

Because there is no Bluetooth SIG official beaconing standard, all beaconing implementations are pseudo-standards, implemented by one company or group of companies and adopted by other companies who want to use them. A short overview of the leading pseudo-standards is provided below with more detailed information on unique packet structures in the Appendix.

Apple’s iBeacon

Apple was an early beaconing adopter with its iBeacon. The iBeacon term is trademarked by Apple, and vendors who want to sell iBeacon products or use the iBeacon logo must obtain a free license from Apple. The iBeacon specification and other developer resources can be downloaded from the Apple Developer website.

Apple iBeacon logo

iBeacon specifies a 30 byte packet which must be broadcast on 100 ms intervals (although iBeacon OEMs don’t always seem to adhere strictly to the 100 ms requirement). The full iBeacon packet structure is shown and explained in detail in the Appendix.

iOS Apps which use the Core Location framework can ask the iOS to continuously monitor for beacon-region-crossing events, i.e., entering or exiting the proximity of an iBeacon defined by the UUID, Major and Minor fields. The iOS monitoring takes place whether the app is running or not, and can even trigger a closed app to launch. Monitoring only works when the user has enabled Location Services for the corresponding app.

Google’s Eddystone

Eddystone is an open-source, cross-platform beacon format from Google. It supports both Android and iOS devices. Unlike other beacon standards, it defines several different frame types which can be used individually or in combination:

- Eddystone-UID which broadcasts a unique beacon ID;

- Eddystone-TLM which can be used to broadcast telemetry (health and status) data about the beacon itself; and
- Eddystone-URL which broadcasts Uniform Resource Locators (URLs).

The Eddystone-URL frame enables mobile platforms to offer web content based on proximity without requiring an app to be installed, enabling what Google has dubbed The Physical Web, or the “ability to walk up and use anything.”

Eddystone logo

Eddystone is already supported in Chrome for iOS, and will be supported in Chrome for Android beginning with version 49. With the Chrome Today widget, users can access web content relevant to their surroundings, and receive notifications when encountering beacons.

Google’s Physical Web uses Eddystone-URL beacons to allow iOS and Android users to “walk up and use anything” in their beacon-tagged surroundings.

The Google Eddystone GitHub page provides the Eddystone Protocol Specification along with tools and open source code examples, and the Google Developers forum provides more information about the Google beacon platform. The full Eddystone beacon format is described in detail in the Appendix.

AltBeacon

Radius Networks defined the AltBeacon specification in an attempt to create an OS-agnostic, open-source standard which wouldn’t favor any particular vendor. The specification is available on the AltBeacon website and is free to use without royalties or licensing fees. Like other beacons, it uses non-connectable, undirected advertising packets. The full packet format is described in the Appendix.

AltBeacon logo

Considerations for Designing a Beacon Product

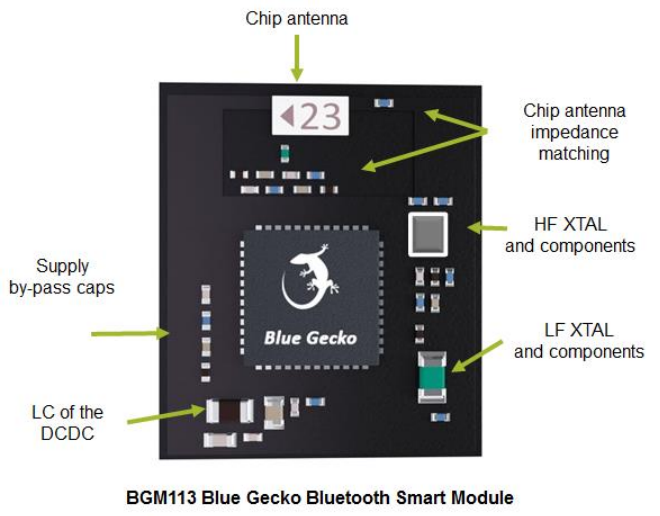
Beacons enable new business models in everything from vending machines to snow-blowers and weed-eaters. Many OEMs who previously never used wireless technology are now adopting Bluetooth and adding beacons to their products.

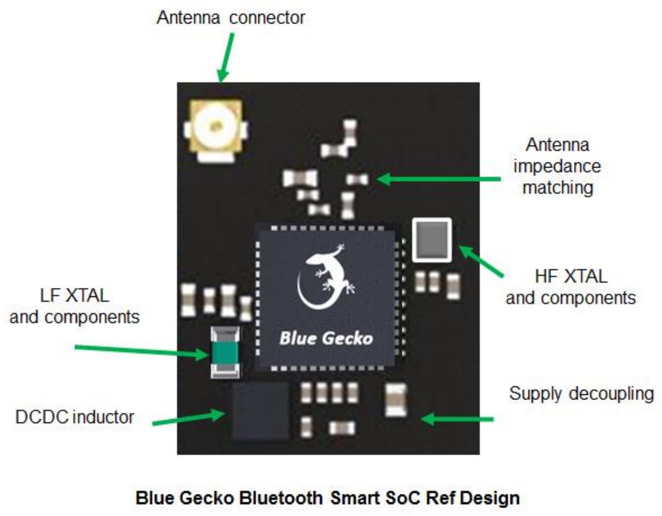
Although a beacon can be relatively simple, the designer will need to take into consideration the hardware and software building blocks, trade-offs with battery life, how the beacon will be provisioned or deployed into the field, security and privacy, and of course their vendor's development tools and support.

Hardware Building Blocks

In its most basic form, a beacon can be implemented with a wireless System-on-Chip (SoC) or a module, along with a battery and a mechanical enclosure. More typically a beacon will include other components that provide functional user interaction such as pushbuttons, LEDs, piezo buzzers, and/or reed switches. If a beacon is embedded into a product with other features, it might include additional sensors such as temperature, light or hall-effect sensors, which can interface directly to the wireless SoC.

A pre-certified module provides the fastest time to market, avoiding significant up-front engineering investments and RF compliance testing, while a discrete SoC design might provide size or cost savings.





Typical pre-certified Bluetooth low energy beaconing module and Bluetooth SoC reference design

The whitepaper Six Hidden Costs in a 99 Cent SoC provides a good discussion of the trade-offs to consider when choosing a module or SoC for a Bluetooth beacon.

Software

It’s fundamentally important to choose a widely deployed and field-proven Bluetooth stack. Most often, this market success is more important than any promises of new cutting edge features. Market success indicates good customer support and a stable stack, both of which help get to market quickly.

For beacons in particular, it’s very important that the protocol efficiently manages sleep modes. When a beacon broadcasts on 100 ms intervals, it only transmits for about 1 ms and sleeps for the other 99; thus spending 99% of its life in sleep mode.

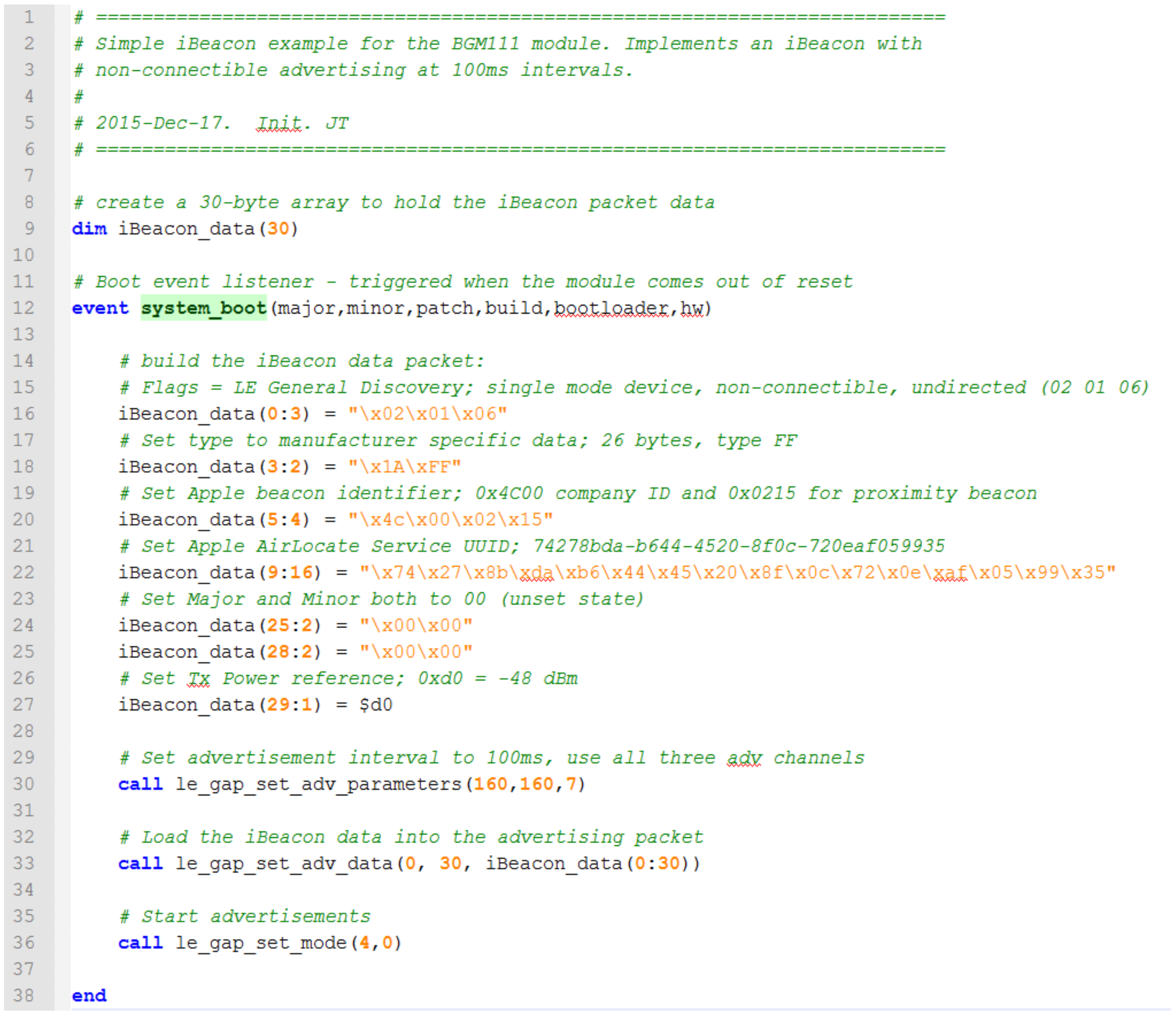
Some proven protocol stacks include special features designed specifically to make Bluetooth and beacons more power efficient. One example is the ability to define multiple beacon frame types (iBeacon, Eddystone-URL, etc.) along with their timing parameters which the stack can then interleave autonomously without the more power-hungry application code running.

Other important software features include watchdog timers as a fail-safe mechanism, real-time clocks to set beacon on/off cycles to preserve power, and the ability to support firmware updates.

Beacon Application Code

Beacon application code can be relatively simple and implemented with a high level programming language like BGScript® from Silicon Labs. BGScript is a simple, high-level, BASIC-like programming language that’s human readable and allows Silicon Labs devices to run standalone without an external processor.

The benefit of this type of high-level programming language is that the developer can focus on the application itself without concerning himself/herself with the timing and complexities of the protocol stack underneath. See below for a simple commented code example for implementing an iBeacon in BGScript.



```
# ===========================================================================
# Simple iBeacon example for the BGM111 module. Implements an iBeacon with
# non-connectible advertising at 100ms intervals.
#
# 2015-Dec-17.  Init. JT
# ===========================================================================

# create a 30-byte array to hold the iBeacon packet data
dim iBeacon_data(30)

# Boot event listener - triggered when the module comes out of reset
event system_boot(major,minor,patch,build,bootloader,hw)

    # build the iBeacon data packet:
    # Flags = LE General Discovery; single mode device, non-connectible, undirected (02 01 06)
    iBeacon_data(0:3) = "\x02\x01\x06"
    # Set type to manufacturer specific data; 26 bytes, type FF
    iBeacon_data(3:2) = "\x1A\xFF"
    # Set Apple beacon identifier; 0x4C00 company ID and 0x0215 for proximity beacon
    iBeacon_data(5:4) = "\x4c\x00\x02\x15"
    # Set Apple AirLocate Service UUID; 74278bda-b644-4520-8f0c-720eaf059935
    iBeacon_data(9:16) = "\x74\x27\x8b\xda\xb6\x44\x45\x20\x8f\x0c\x72\x0e\xaf\x05\x99\x35"
    # Set Major and Minor both to 00 (unset state)
    iBeacon_data(25:2) = "\x00\x00"
    iBeacon_data(28:2) = "\x00\x00"
    # Set Tx Power reference; 0xd0 = -48 dBm
    iBeacon_data(29:1) = $d0

    # Set advertisement interval to 100ms, use all three adv channels
    call le_gap_set_adv_parameters(160,160,7)

    # Load the iBeacon data into the advertising packet
    call le_gap_set_adv_data(0, 30, iBeacon_data(0:30))

    # Start advertisements
    call le_gap_set_mode(4,0)

end
```

BGScript iBeacon example code for the BGM111 Bluetooth low energy module

This example code supports the BGM111 module and is implemented with only 38 lines (most of it comments). The code's advertising packet is constructed to use the Apple AirLocate Service UUID of 74278bda-b644-4520-8f0c-720eaf059935, with major and minor fields of 0x00, meaning they are unset. The calibrated Tx Power value of 0xD0 correlates to -48 dBm at one meter. (The BGM111 has +8 dBm of Tx power.)

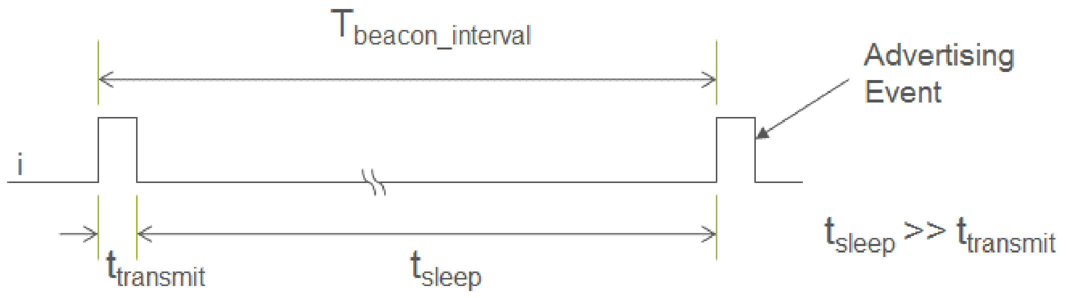
This is a very basic example of an iBeacon implementation, but a more robust implementation might include the ability to change the beacon's UUID and major/minor values through the over-the-air interface, and possibly include provisions for updating the beacon's firmware.

Battery Life

As with any product, the size of the battery versus its power consumption will determine its operating life. Battery size is sometimes dictated by the beacon industrial design, so a compromise must be made between battery life and the physical constraints of the final product.

The beacon's transmit power and beaconing interval also play an important role in battery life. These parameters are typically trade-offs against the desired range and proximity accuracy. Higher transmit power provides longer range and a wider coverage area, but the transmitter will draw more power from the battery for each beacon event. The application usage model needs to be considered: Will the application benefit from a larger coverage area? Or is it better to have its proximity limited to a few square meters?

Likewise, the beaconing interval is an important factor in determining a beacon's battery life. A beacon's power consumption peaks at several milli-amps while transmitting. Otherwise, it only draws micro-amps while in deep sleep.



$$I_{avg} = \frac{i_{transmit}t_{transmit} + i_{sleep}t_{sleep}}{T_{beacon\_interval}}$$

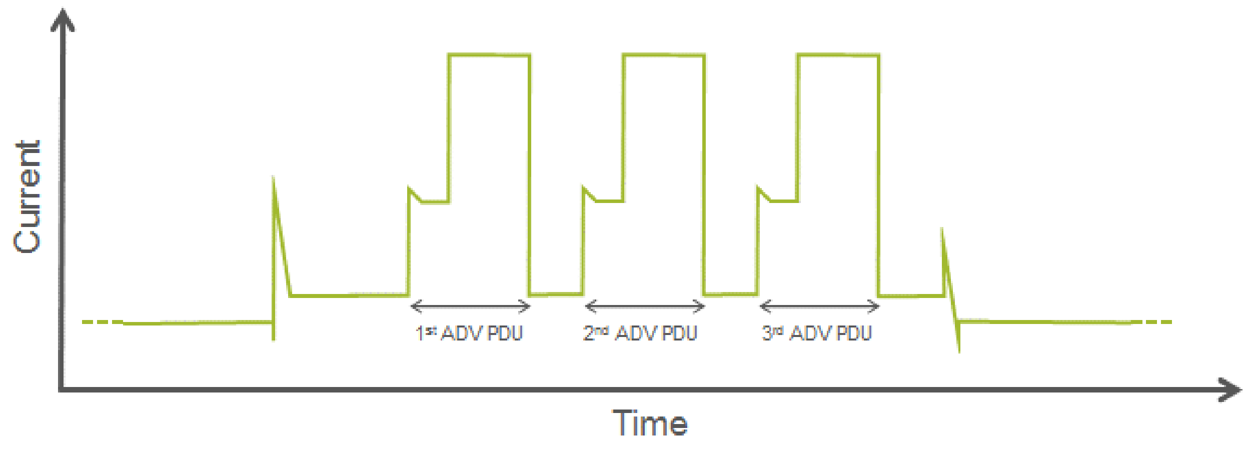
A beacon's average battery life is determined by transmit power and its transmit / sleep duty cycle

There are important trade-offs for short versus long transmit cycles, including operating life and location accuracy.

With transmit power constant, short cycles provide more data points for better location, while longer cycles provide worse location accuracy but longer battery life.

In terms of performance, a shorter beaconing interval means there are more beacon events to capture, providing more motion resolution and therefore better location accuracy. A longer interval means the beacon will have longer battery life but fewer opportunities to get captured, especially by a moving smartphone.

Beacon interval also plays a role in smartphone application response times. When apps are in the background, the OS scans for beacons less frequently, so a more frequent beacon interval increases the likelihood of detection and resulting activity.



Current profile in a typical Bluetooth low energy advertising event (ADV PDU)

Privacy and Security Issues

Recent press coverage has highlighted fears among some consumers about beacons "tracking your every move." In reality, typical beacons do not collect data since they are one-way devices—they only broadcast.

They do however, provide the ability for a smartphone to know when it's near a known beacon, and in some cases (if the beacon is stationary), the smartphone might have access to information about the location of that beacon. The smartphone translates this information to provide location and usage-based services, either through an app or in the case of the Physical Web, with contextually-relevant search results.

It's important to note that the same smartphone can provide the same services based on GPS, Wi-Fi, or cell towers, so beacons are not exposing new concerns, just making them more widespread through their success. In all cases, smartphone users can simply enable or disable proximity services in their settings.

Smartphones can use GPS, Wi-Fi, cellular, and Bluetooth beacons for proximity-aware applications.

And users can disable them all in settings.

Similarly, concerns have been raised about beacons creating added security risks for IT systems. But this again implies capabilities that most beacons do not have. The typical beacon is a standalone device with no connection to any other network, wired or wireless.

However, some beacons are designed with infrastructure network access which allows for central management of a beacon fleet. In these cases, the beacon manufacturer provides the same level of tamper security as they would for any other device attached to their IT network. On the beaconing side, the beacon's data is, by design, broadcast for all to hear so does not necessarily need to be encrypted or protected in any way.

In beaconing applications where proximity to the beacon may have tangible value, such as reward points for example, the beacon OEM will implement additional safeguards against beacon spoofing. Without them, spoofed beacons might fool the system into crediting events too often or to the wrong person. The safeguards could include simple timestamps for each proximity event with a test for an improbable or unanticipated frequency, the use of ephemeral IDs, or the use of randomized security keys, generated with each proximity event and validated by the back-end system.

Security for Device Management Functions

In general, Bluetooth security isn't a concern with beacons since by definition they are broadcast devices with the intent that any listener can receive the data.

But if the device includes other internal services like device management functions, then those services might be protected and require authentication before allowing access. In those cases, the beacon will use the security features built into the Bluetooth protocol (i.e., pairing, authentication, encryption, etc.) and other security measures implemented by the beacon OEM such as strong password protection.

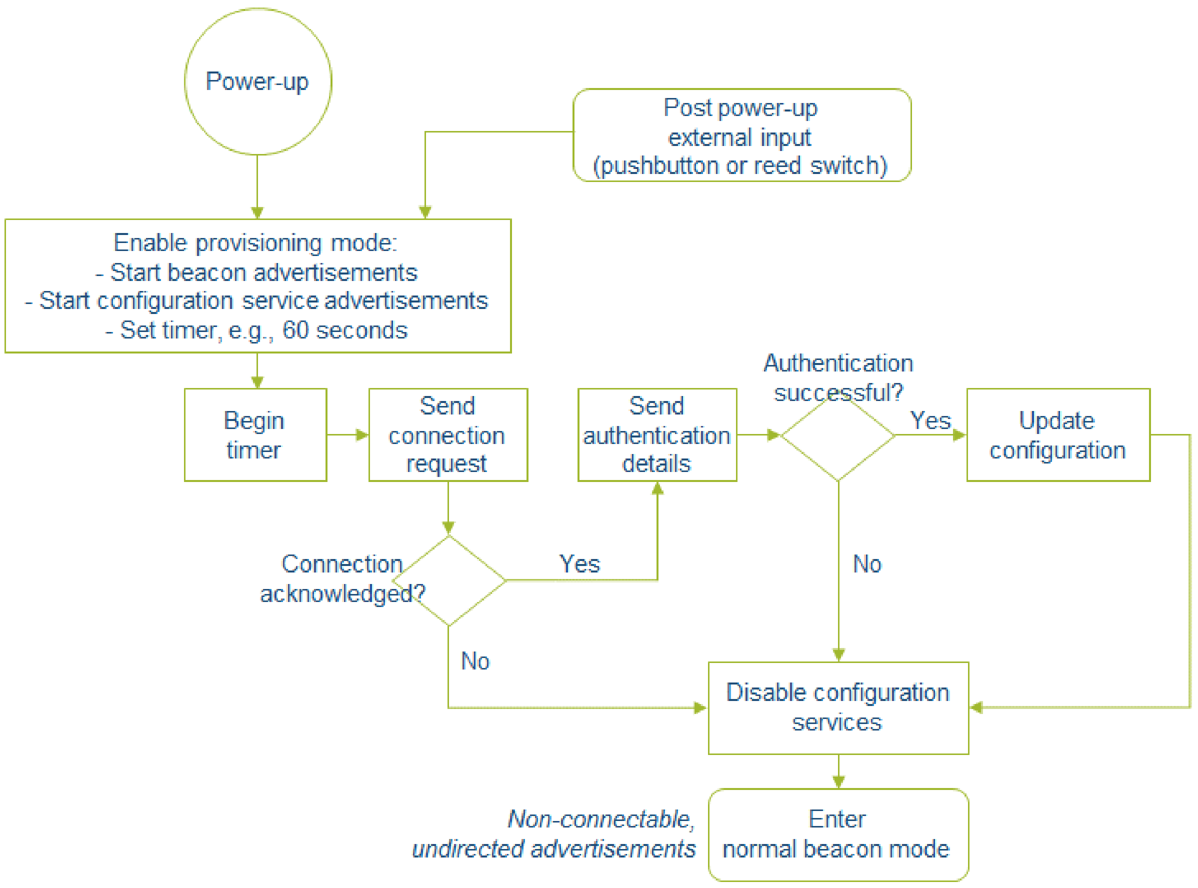
The most common device management functions are used to configure the beacon during the provisioning, and access to those services may be limited to a short window of time after the device is reset. After the access window expires the device becomes a normal broadcast beacon and no longer advertises its internal services.

Provisioning / Deploying the Beacon

When beacons are deployed in the field, it's often necessary for the installer to configure the beacon to the installation, for example by writing a unique major/minor value or instance ID. The most obvious (and most desirable) way to do this is to take advantage of the over-the-air interface. But exposing the configuration services over Bluetooth can provide a potential channel for unauthorized access, and the extra advertising packets for the configuration services can have a significant impact on battery life.

A simple, commonly used solution is to limit exposure of these services to a short window of time either just after power-up, or after a trigger from a physical input such as a pushbutton or reed switch. This is illustrated below.

During the configuration window, the beacon is connectable (with appropriate authentication) so that the provisioning configuration can be written into it. Once the timer expires, it reverts to only advertising the non-connectable beacon packets, effectively hiding the configuration services.


Power-up
Post power-up external input (pushbutton or reed switch)
Enable provisioning mode:
- Start beacon advertisements
- Start configuration service advertisements
- Set timer, e.g., 60 seconds
Begin timer
Send connection request
Send authentication details
Authentication successful?
Yes
Update configuration
Connection acknowledged?
Yes
No
No
Disable configuration services
Non-connectable, undirected advertisements
Enter normal beacon mode

Flowchart for a time-limited provisioning window

Summary

It's hard to imagine we all won't be touched by beacon applications in the near future. Indeed, they may be the next killer app. Product designers across hundreds of industries will have lots of new challenges to address as they move to adopt wireless. Choosing the right vendor with innovative technology, a market-proven stack, and great customer support will help ensure the developer has a smooth experience and a superior final product.

Silicon Labs is an established leader in Bluetooth low energy and beaconing technology, development tools, and a broad portfolio of Bluetooth beacon-capable SoCs and pre-certified Bluetooth modules.

Get Started here with Blue Gecko Bluetooth low energy modules and SoCs.

Appendix 1 – Example Code and Further Reading

Code examples for various beacon implementations are provided on the Silicon Labs website, including an iBeacon example with parameters that are configurable over the air, or through a wired host interface, and an example which implements interleaved iBeacon and Eddystone-URL beacons in the same device. All of the examples are written in BGScript code and provided as full project files with source code, which can run on any Silicon Labs Bluetooth product:

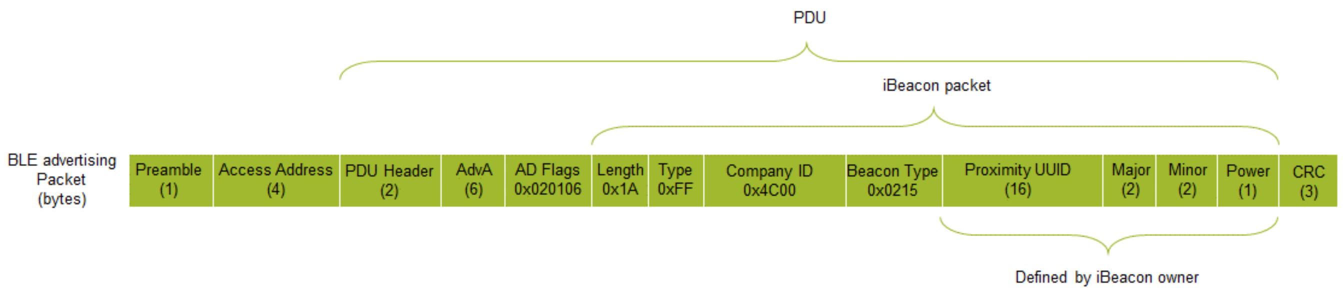
- Video: Introduction to Bluetooth low energy
- Video: Blue Gecko Getting Started
- Application Guide: UG103.14: Application Development Fundamentals for Bluetooth Smart
- Example Code: dual beacon for BLExxx
- Example Code: Physical Web beacon using BLED112 USB dongle
- Community Topic: Bluetooth low energy data transfer basics
- Community Topic: Bonding, Encryption and MITM protection

Appendix 2 – Detailed Frame Structure and Explanation of iBeacon, Eddystone, and AltBeacon

Apple's iBeacon

Apple was an early adopter of beaconing and calls their implementation iBeacon. The iBeacon term is trademarked by Apple, and vendors who want to sell iBeacon products or use the iBeacon logo must obtain a free license from Apple. The iBeacon specification and other developer resources can be downloaded from the Apple Developer website.

iBeacon specifies a 30 byte packet which must be broadcast on 100 millisecond intervals (although iBeacon OEMs don't always seem to adhere strictly to the 100 ms requirement). The packet structure is shown in Figure X and explained below.



Apple iBeacon advertising packet

Flags: These first three bytes follow the [length, type, data] structure explained above, and indicate:

Length 0x02 = 2 bytes

Type 0x01 = flags

Value 0x06 = (non-connectable, undirected advertising, single-mode device)

Beacon Identifier: The remainder of the packet is the beacon identifier:

Length 0x1A = 26 bytes (the remainder of the packet)

Type 0xFF = manufacturer specific data

Value = the manufacturer specific data is further sub-divided into the following fields:

Company ID: Two bytes always set to 0x4C00

Beacon Type: Two bytes always set to 0x0215 for proximity beacons

Proximity UUID: A sixteen byte (128 bit) UUID set by the beacon owner or set to one of several Apple AirLocate Service UUID values.

Major: Two bytes which can be used to define a sub-region within the larger region defined by the UUID. A common example of using the Major field would be to use it for designating a particular store for a retail chain. A value of 0x0000 means the Major value has not been set.

Minor: Two bytes which can be used to a further subdivide the region defined by the Major field. A common example of using the Minor field would be to use it for designating a particular department inside the store designated by the Major field. A value of 0x0000 means the Minor value has not been set.

Measured Power: A one byte value indicating the iBeacon's calibrated output power in dBm measured at a distance of 1 meter. This allows the iOS device to improve its ranging accuracy based on the Received Signal Strength Indicator (RSSI) for received iBeacon packets. The value is a signed 8-bit integer, and since the received power level is typically smaller than 1 milliwatt the dBm value is negative. Thus it is indicated by a twos-complement number. For example,

RSSI = -60 dBm; Measured Power = 256 – 60 = 196 = 0xC4

iOS apps which use the Core Location framework can ask the operating system to continuously monitor for region-crossing events, i.e., entering or exiting the proximity of an iBeacon defined by the UUID, Major and Minor fields. The monitoring takes place whether the app is running or not, and can even trigger an app to launch if it is closed. Monitoring only works when the user has enabled Location Services for the corresponding app.

Google's Eddystone

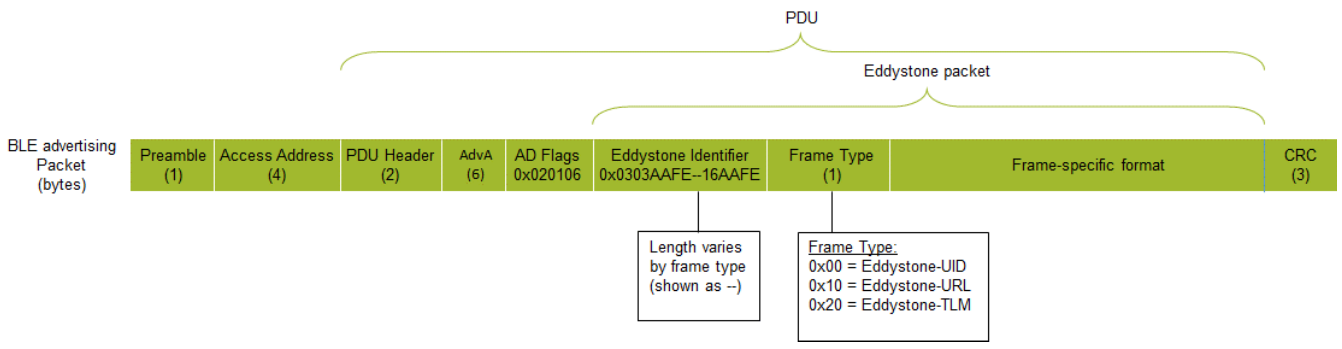
Eddystone is an open-source, cross-platform format from Google for proximity beacons that supports both Android and iOS devices.

Unlike other beacons, it defines several different frame types which can be used individually or in combination:

- Eddystone-UID which broadcasts a unique beacon ID;
- Eddystone-URL which broadcasts Uniform Resource Locators (URLs); and
- Eddystone-TLM which can be used to broadcast telemetry (health and status) data about the beacon itself.

- Eddystone-EID which uses ephemeral (short lived) identifiers for beacon applications requiring more security. The specification for this frame format has not yet been released.

The Eddystone-URL frame enables mobile platforms to offer web content based on proximity without requiring an app to be installed first, enabling what Google has dubbed The Physical Web. The Eddystone Protocol Specification along with tools and open source code examples are on Google's Eddystone GitHub page. The Eddystone beacon format is shown below with each frame type described.



Eddystone basic packet

Flags: These first three bytes follow the [length, type, data] structure explained above, and indicate:

Length 0x02 = 2 bytes

Type 0x01 = flags

Value 0x06 = (non-connectable, undirected advertising, single-mode device)

Beacon Identifier: The following eight bytes identify the beacon as Eddystone:

Length 0x03 = three bytes (one for type, two for value)

Type 0x03 = the type descriptor for *Complete List of 16-bit Service Class UUIDs*

Value 0xAAFE = the 16-bit Eddystone Service UUID, always 0xAAFE

Length 0x-- = the length varies depending on frame type and its data

Type 0x16 = the type descriptor for *Service Data*

Value 0xAAFE= the 16-bit Eddystone Service UUID, always 0xAAFE

Frame-specific Service Data: The remainder of the advertising packet contains one or more of the individual Eddystone frame types. The frame type is identified by the first byte of the Service Data.

Eddystone-UID: The Eddystone-UID frame type is a 16-byte unique beacon ID broken into a 10-byte namespace identifier and a 6-byte instance identifier, both assigned by the beacon owner. The frame length is fixed and uses the entire advertising packet, so in this case the Service Data length byte should be 0x17.

Frame type 0x00 = Eddystone-UID frame type

Tx power (one byte) = a signed 8-bit integer indicating the beacon's Tx power in dBm measured at 0 meters. This can be calculated by measuring Tx power at 1 meter and adding 41 dBm.

NID = ten byte namespace ID. To ensure uniqueness, the recommended format for the namespace identifier is to use either SHA-1 hash of a qualified domain name, or an elided UUID with bytes 5-10 removed.

BID = six byte instance ID which does not need to adhere to any specific format.

RFU = two bytes reserved for future use

Eddystone-URL: The Eddystone-URL frame type advertises a URL using a compressed format for internet resources accessible with hyper-text transfer protocol (HTTP or HTTPS).

Frame type 0x10 = Eddystone-URL frame type

Tx power (one byte) = a signed 8-bit integer indicating the beacon's Tx power in dBm measured at 0 meters. This can be calculated by measuring Tx power at 1 meter and adding 41 dBm.

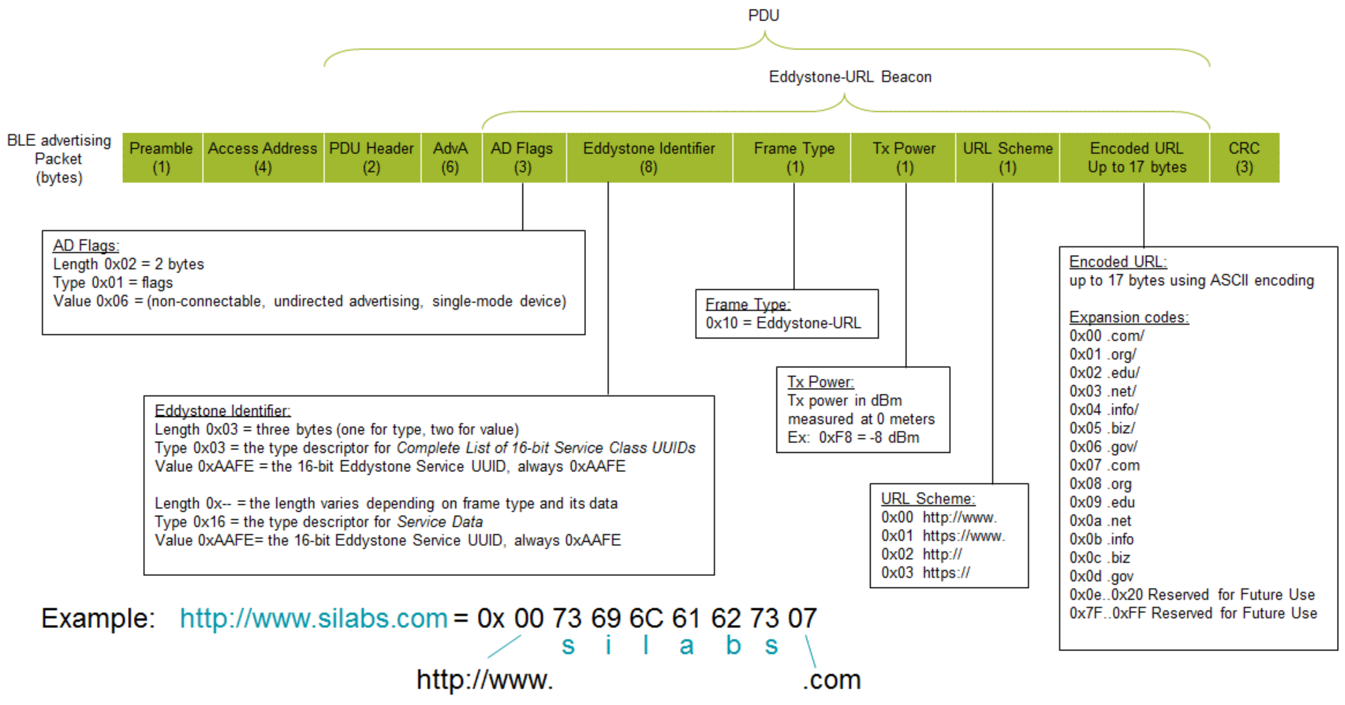
URL scheme = one byte which designates the URL prefix. For example 0x00 = "http://www".

Encoded URL = up to 17 bytes for the URL using ASCII encoding. If character codes are used which are not allowed by IETF RFC 1738, then they are interpreted as expansion codes. For example 0x00 is an invalid character code, and gets translated in the Eddystone-URL beacon as ".com/". Using the compressed format for the URL http://www.silabs.com would result in:

00 73 69 6c 61 62 73 00

For the full list of prefixes and expansion codes which can be used, refer to the Eddystone-URL spec on the Eddystone GitHub page. They are also shown in the detailed chart below.

To shorten a URL which is longer than 17 bytes, there are many several URL shortening services on the web such as goo.gl and bit.ly.


PDU
Eddystone-URL Beacon
BLE advertising Packet (bytes)
Preamble (1)
Access Address (4)
PDU Header (2)
AdvA (6)
AD Flags (3)
Eddystone Identifier (8)
Frame Type (1)
Tx Power (1)
URL Scheme (1)
Encoded URL Up to 17 bytes
CRC (3)
AD Flags:
Length 0x02 = 2 bytes
Type 0x01 = flags
Value 0x06 = (non-connectable, undirected advertising, single-mode device)
Eddystone Identifier:
Length 0x03 = three bytes (one for type, two for value)
Type 0x03 = the type descriptor for Complete List of 16-bit Service Class UUIDs
Value 0xAAFE = the 16-bit Eddystone Service UUID, always 0xAAFE
Length 0x-- = the length varies depending on frame type and its data
Type 0x16 = the type descriptor for Service Data
Value 0xAAFE= the 16-bit Eddystone Service UUID, always 0xAAFE
Frame Type:
0x10 = Eddystone-URL
Tx Power:
Tx power in dBm measured at 0 meters
Ex: 0xF8 = -8 dBm
URL Scheme:
0x00 http://www.
0x01 https://www.
0x02 http://
0x03 https://
Encoded URL:
up to 17 bytes using ASCII encoding
Expansion codes:
0x00 .com/
0x01 .org/
0x02 .edu/
0x03 .net/
0x04 .info/
0x05 .biz/
0x06 .gov/
0x07 .com
0x08 .org
0x09 .edu
0x0a .net
0x0b .info
0x0c .biz
0x0d .gov
0x0e..0x20 Reserved for Future Use
0x7F..0xFF Reserved for Future Use
Example: http://www.silabs.com = 0x 00 73 69 6C 61 62 73 07
s i l a b s
http://www.
.com

Eddystone-URL packet

Eddystone-TLM: The Eddystone-TLM frame type allows the beacon to broadcast information about its own internal operations, which can be used to monitor the health of the beacon. Since TLM frames don't contain any identifying information they must always be interleaved with either a URL or UID frame to enable association to the correct beacon. The TLM frame has a fixed length so the Service Data length byte should always be 0x11.

Frame type 0x20 = Eddystone-TLM frame type

TLM version 0x00 = one byte indicating the version. This is to allow future enhancements and should always be set to 0x00.

VBATT = two bytes indicating the battery voltage with a resolution of 1 mV/bit. If not supported, the value should be 0x0000

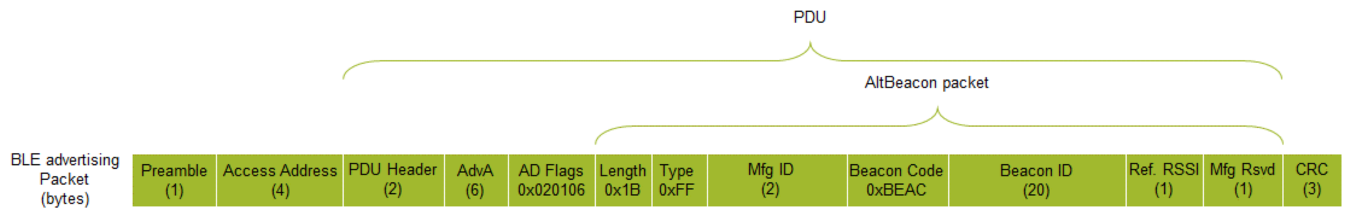
TEMP = two bytes indicating the beacon temperature in degrees C, and represented using 8:8 fixed point notation (the first byte indicating the signed integer portion and the second byte indicating the fractional portion). For example, 1.5 °C is represented as 0x0180 and -1.5 °C is represented as 0xFE80. If not supported the TEMP value should be set to 0x8000.

ADV_CNT = four bytes indicating the number of advertising events (of all frame types) since the last reset.

SEC_CNT = four bytes indicating the elapsed time since the last reset in 0.1 second increments. A four byte value provides enough scope to show 13.5 years of continuous operation.

AltBeacon

The AltBeacon specification was defined by Radius Networks in an attempt to create an open-source, OS-agnostic standard for proximity beacons which wouldn't favor any particular vendor. The specification is available on the AltBeacon website at www.altbeacon.org and is free to use without royalties or licensing fees. Like other beacons, it uses non-connectable, undirected advertising packets. The packet format is shown and described below. The current specification does not define a particular beacon timing interval.



AltBeacon Packet

Flags: These first three bytes follow the [length, type, data] structure explained above, and indicate:

Length 0x02 = 2 bytes

Type 0x01 = flags

Value 0x06 = (non-connectable, undirected advertising, single-mode device)

Beacon Identifier: The remainder of the packet is the beacon identifier:

Length 0x1B = 27 bytes (the remainder of the packet)

Type 0xFF = manufacturer specific data

Value = the manufacturer specific data is further sub-divided into the following fields:

Mfg ID: Two bytes which define the beacon manufacturer’s identifier from the Bluetooth SIG assigned numbers database.

Beacon Code: Two bytes always set to 0xBEAC

Beacon ID: A twenty byte value which uniquely identifies the beacon. The first sixteen bytes of this field should uniquely identify the organization

Reference RSSI: A signed one byte value indicating average received signal strength at 1m from the beacon. The range is from 0 to -127.

Mfg Reserved: A one byte value reserved for use that is defined by the beacon manufacturer.

Additional References

Specification of the Bluetooth System, Covered Core Package version: 4.2 (2014, Dec 2). Retrieved from https://www.bluetooth.com/specifications/adopted-specifications.

Bluetooth Core Specification Addendum 5 (2015, Dec 15). Retrieved from https://www.bluetooth.com/specifications/adopted-specifications.

Gast, M. (2014). *Building Applications with iBeacon*, Sebasopol, California: O'Reilly Media, Inc.

Statler, S. (2016). *Beacon Technologies: The Hitchhiker's Guide to the Beacosystem*, New York, NY: Apress Media LLC.

Heydon, R. (2012). *Bluetooth Low Energy – The Developers Handbook*, Upper Saddle River, New Jersey: Pearson Education, Inc.

Proximity Beacon Specification R1 (2015, Sep 4). Retrieved from https://developer.apple.com/ibeacon.

Eddystone Protocol Specification. (n.d.) Retrieved from https://github.com/google/eddystone/blob/master/protocol-specification.md.

AltBeacon Protocol Specification v1.0. (n.d.) Retrieved from https://github.com/AltBeacon/spec

Bluetooth SIG (2015) – Bluetooth Smart Technology: Powering the Internet of Things – https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy

Six Hidden Costs in a 99 Cent Wireless SoC (2015). Retrieved from http://www.silabs.com/Support%20Documents/TechnicalDocs/six-hidden-costs-of-a-99-cent-soc.pdf