MASTERING THE MICROWEB, Bluetooth Low Energy

References

- The Horse's mouth: http://www.bluetooth.com/Pages/low-energy.aspx
- The Bluetooth SIG has a substantial slide deck, search for BLE_101.pdf
- Developer's overview: https://developer.bluetooth.org/DevelopmentResources/Pages/Getting-Started.aspx
- WWDC 2012, 2013 CoreBluetooth sessions: https://developer.apple.com/videos/
- Apple Sample Code: https://developer.apple.com/search/index.php?q=bluetooth%2Ble
- IEEE, comparison of LE vs ZigBee: http://chapters.comsoc.org/vancouver/btler3.pdf
- ConnectBlue White Papers are very good: http://support.connectblue.com
- Cool tools for prototyping LE server hardware: http://bleduino.cc

How is Bluetooth Low Energy different?

- Lower radio power, ~10mW versus ~100mW for classic BT
- Radio redesign for very short duty cycles
- Smaller Packets (typical 20 byte payloads)
- Not intended for data streaming, effective rate is only ~1Mbps
- Transaction based, most traffic can be connectionless.
- Generic Attribute (GATT) profiles simplify cross-compatibility
- Very low cost chipsets.

Radio Requirements:

- Bluetooth 4.0, Low Energy requires new hardware.
- Single mode devices, usually servers and peripherals, only support BT LE
- Dual mode chipsets support Classic and LE, and are usually in central clients.
- Dual Mode: iPhone4s, iPad3 & Mini, most 2012+ Macs (except Pro)
- Bluetooth LE USB dongles are common and cheap, results vary wildly.
- Very little (no?) support in Android hardware.

Spectrum and the Advertising Model

- 2.4 GHz, GFSK (Gaussian frequency-shift keying) in 40 channels at 2MHz spacing
- Advertising channels tagged 37, 38, & 39, but are scattered to 2402, 2426, & 2480 MHz
- Advertising is analogous to a Sonar ping and the subsequent pause for a response.

Are you being Served?

- BT LE defines a server as a supplier of data, e.g. instrument package, AKA a peripheral.
- Servers advertise (broadcast) their availability, capabilities, and optionally current data.
- Clients, AKA Central Managers, consume server data broadcasts, optionally connecting.
- A client, the data consumer, is a PC, Mac, iPhone, cloud data aggregator, etc.
- iOS apps may act as data clients (Central Managers) or data servers (Peripherals)
- Pairing, encryption, and stored pairing (bonding) are available but often superfluous.
- On pairing, the Central Manager 'owns' the peripheral.

Consuming Data, Classes for Central Managers

- Event driven, your manager objects will be delegates for the Core Bluetooth objects.

- CBCentralManagerDelegate: Discover, Connect, and Reconnect to peripherals.
  Vaguely analogous to filesystem find, open, and reopen mechanisms.
  Use when you wish to connect to a server, represented as a CBPeripheral.

- CBPeripheralDelegate: Enumerate peripheral services, set and get peripheral state.
  much like keyboard and mouse i/o. In transitional code HID often used as an API.

- CBService, CBMutableService: Container, getter/setter for a single peripheral service.
  Fine grained, handles a single element of the several a peripheral might supply.

Hooking to bLE, the CBCentralManagerDelegate

```
@interface
  LErkCloud:NSObject<CBCentralManagerDelegate,CBPeripheralDelegate>
{}
```

```
...
self.manager = [[CBCentralManager alloc] initWithDelegate:self
                                    queue:LErkCloudQueue()];
...
```

Finding a Service, your CBCentralManager seeks volunteers

```
-(void)centralManagerDidUpdateState:(CBCentralManager *)central
{
    self.bLEready = [self isLECapable];
}

- (void) startScan
{
    if(self.bLEready) {
        dispatch_async(LErkCloudQueue(), ^{
            [self.manager
              scanForPeripheralsWithServices:myServicesDict
                                     options:nil];
        });
    } else {
        NSLog(@"No scan for you! blueToothLE not ready.\n");
    }
}
```

Devices enumerated via CBCentralManager delegate methods

```
-(void)centralManager:(CBCentralManager *)central
    didDiscoverPeripheral:(CBPeripheral *)peripheral
    advertisementData:(NSDictionary *)advertisementData
    RSSI:(NSNumber *)RSSI
{...}

-(void)centralManager:(CBCentralManager *)central
    didRetrievePeripherals:(NSArray *)peripherals
{...}
```

CBCentralManagerDelegate, attempting to connect

```
    ...
    dispatch_async(LErkCloudQueue(), ^{
        [self.manager connectPeripheral:peripheral options:nil];
    });
    ...

-(void)centralManager:(CBCentralManager *)central
        didConnectPeripheral:(CBPeripheral *)aPeripheral
{...}


-(void)centralManager:(CBCentralManager *)central
        didFailToConnectPeripheral:(CBPeripheral *)aPeripheral
        error:(NSError *)error
{...}
```

Catch server chatter via CBPeripheralDelegate classes

```
/* completion of a -[discoverServices:] request. */
- (void)      peripheral:(CBPeripheral *)aPeripheral
    didDiscoverServices:(NSError *)error
{
     NSLog(@"Device %@ Services:\n",[aPeripheral name]);
     for (CBService *aService in aPeripheral.services)
     {
          if([aService includedServices]) {
               dispatch_async(LErkCloudQueue(), ^{
                    [aPeripheral discoverIncludedServices:nil
                                               forService:aService];
               });
...
```

Discover server capabilities via CBPeripheralDelegate

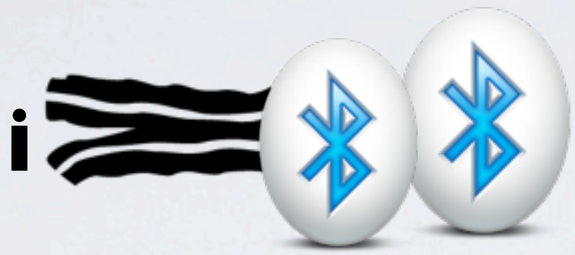
```
// completion of a [discoverCharacteristics:forService:] request.
- (void) peripheral:(CBPeripheral *)aPeripheral
            didDiscoverCharacteristicsForService:(CBService *)service
                                           error:(NSError *)error
{
    dispatch_async(LErkCloudQueue(), ^{
        for (CBCharacteristic *aChar in service.characteristics)
        {
            [aPeripheral setNotifyValue:true
                      forCharacteristic:aChar];
        }
    });
}
```

Catch server state changes via CBPeripheralDelegate

```
// completion of a -[readValueForCharacteristic:] request
// or the reception of a notification/indication.
- (void) peripheral:(CBPeripheral *)aPeripheral
     didUpdateValueForCharacteristic:(CBCharacteristic *)characteristic
                               error:(NSError *)error
{
    dispatch_async(LErkCloudQueue(), ^{
        // queue up a useful response to the message
        [self updateEventCandidate:characteristic
                      forPeripheral:aPeripheral];
    });
}
```

Obligatory Buzzwordery

- iEggs & iBeacon



- Bumping or Beaconing?
  http://gigaom.com/2013/09/10/with-ibeacon-apple-is-going-to-dump-on-nfc-and-embrace-the-internet-of-things/
- LE broadcast-only servers.
- The internet of things.
- Still no Mesh nets, AKA where are my ZigBees?