



E2E Bluetooth 4.0 Compliant Sensor v1.0

Overview

The E2E sensor (E2ES) is a Bluetooth 4.0-compliant device that logs temperature data. E2ES was designed to be easy to use and to overcome the shortfalls of other similar devices.

E2ES uses a virtual UART-style communication scheme on top of Bluetooth 4.0 services. E2ES logs data in a circular fashion and may make use of its single push-button and green LED for user interaction. E2ES also has provisions for additional sensors such as Light or Vibration.

Bluetooth Attributes

E2ES design is based on the Nordic nRF52832 ARM based device with integrated 2.4 GHz radio which exhibits the following characteristics:

- Advertise Interval 1 - 2 Seconds
- Battery Life ~3 Years

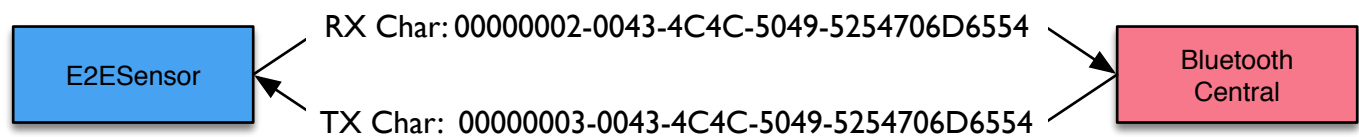
Logger Attributes

E2ES logging is built on top of the Nordic platform but is an abstraction that can be adapted to other radio communication devices. Many of the properties of the logging facilities are governed by the underlying platform and may change in future updates or different hardware. Logging is mildly configurable. All configurable options can be set over Bluetooth while the logger is Stopped.

- Data Point Limit 12,000
- Data Storage Method 32 bit word with 3 10-bit temperatures and 2 bits for mark data
- Temperature Range -50 to +50 Degrees Celsius
- Temperature Accuracy +/- 0.5 Degrees Celsius
- Log Interval 1 to 2^16 Seconds
- Log Delay 1 to 2^16 Seconds

E2ES Interaction

E2ES Interaction begins with a Bluetooth 4.0 scan. All valid E2ES are Generally Discoverable and will advertise a packet containing the name ‘E2ESensor’ in the payload. Once found, issue a Service Discovery to find the transmit and receive services. Communication is now possible by writing data to the transmit channel and reading from the receive channel.



E2ES communication can be done in two ways. The first way is to do a ‘write verify’ on the TX line. A write verify will only return once the RX buffer has been filled and you can safely do a read from the RX channel. The second way would be to write data on the TX channel and await a ‘notify’ event that indicated the RX channel has data available. The first method is well tested and the current recommended approach.

E2ES States

E2ES may be in one of three states:



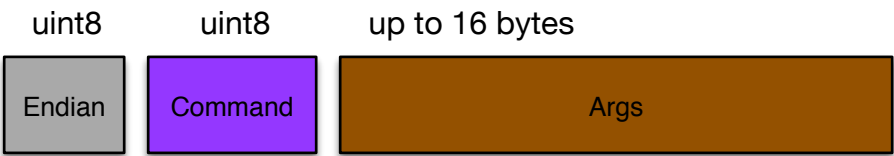
In the Idle state, no data is being logged and there is no radio activity. The only way to remove the E2ES from Idle is to manually push the button.

In the Started state, data is being logged in the circular buffer and the Radio is functioning. The only way to stop the E2ES from logging is to issue the Halt command.

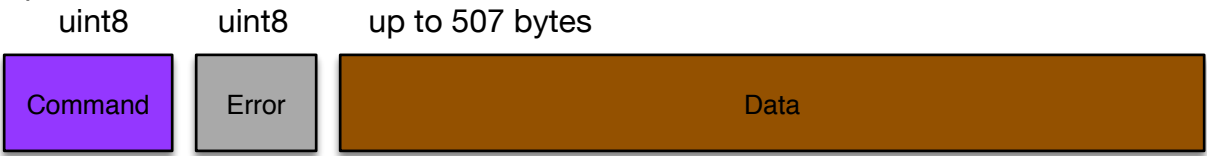
In the Silenced state, data is being logged in the circular buffer and there is no radio activity. The only way to go back to Started state is to manually push the button OR the E2ES will revert to the logging state after X seconds (set when entering Silenced state).

E2ES Commands

Command Format:



Response Format:



Commands take an endian value which dictates the endian of the command and the endian of the response data. The second value is the command type followed by any arguments.

Responses are prefaced by their command type for verification and dispatch. The second value is the error flag and the remaining bytes are any data returned by the command issued.

Endian Values:

0 = Little, 1 = Big

Commands:

- 'I' = Info
- 'T' = Current Temperature
- 'R' = Read Block
- 'U' = Unlock
- 'S' = Silence
- 'H' = Halt
- 'Q' = Quell

Errors:

- 0 = Success
- 1 = Unknown Command
- 2 = Bad Permissions
- 3 = Incorrect Password
- 4 = Unknown Error

Permission Level:

- 0 = Level 0
- 1 = Level 1

States:

- 0 = Idle
- 1 = Started

Discovering Endian of Central system (Example in javascript):

```
function checkEndian() {  
  
    if (endianness === undefined) {  
  
        var a = new ArrayBuffer(4);  
        var c = new Uint32Array(a);  
  
        c[0] = 0x01020304;  
  
        if(b[0] === 0x04) return 0; // Little  
  
        if(b[0] === 0x01) return 1; // Big  
    }  
}
```

Command List

Info Command

Takes no arguments and returns a known response format containing all available information about the connected E2ES.

Ex:

```
TX ->  
0x01      - Endian  
0x49      - Command (Info)  
  
RX <-  
0x49      - Command (Info)  
0x00      - Error (success)  
0x00      - Permission (Level 0)  
0x00      - State (Idle)  
0x00 0x03 - Version (0.3)  
0x5A 0x02 - Power (2.9) - not right, Endian appears backwards  
0x00 0x00 - Data Points Logged (0)  
0x01 0x00 - Data Bytes per Block (256)  
0x00 0xC0 - Data Points per Block (192)  
0x02 0x58 - Log Interval (600)  
0xD8 0x63 0xE3 0x4D 0xA5 0xD2 0xBE 0x01 0xAB 0x48 0x68 0x8D 0x2C 0x5A 0x93 0x61  
- Remaining 16 bytes are the logon challenge
```

Current Temperature Command

Takes no arguments and returns a 16 bit response containing the current temperature reported by the connected E2ES. Value requires conversion - See 'Parsing Temperature Data' below.

Ex:

```
TX ->  
0x01      - Endian  
0x54      - Command (Temperature)  
  
RX <-  
0x54      - Command (Temperature)  
0x00      - Error (success)  
0x02 0x8E - Temperature (15.4 C)
```

Read Block Command

Takes a single uint8 type argument designating the Block to read. Returns a data payload equal in bytes to the 'Data Bytes per Block' value returned by the Info command.

Ex:

```
TX ->
0x01    - Endian
0x52    - Command (Read)
Args... - uint8 value for the sequential block number to read.
```

```
RX <-
0x52    - Command (Read)
0x00    - Error (success)
0x04    - Block Number (4)
```

Unlock Command

Takes a 16 byte argument that must be the calculated challenge response. All commands except the Info command require Unlock Level 1! Current prototypes do not check the response code so you can send any sequence of bytes, for now.

Ex:

```
TX ->
0x01    - Endian
0x55    - Command (Unlock)
0x00    - 16 byte challenge response
....
0x0F
```

```
RX <-
0x55    - Command (Unlock)
0x00    - Error (success)
```

Silence Command

Takes a single uint16 type argument representing the time in seconds the radio should be offline. Passing in a value of 0 will have no effect.

Ex:

```
TX ->
0x01    - Endian
0x53    - Command (Silence)
0x01    - Silence time (0x012C = 300 seconds)
0x2C
```

```
RX <-
0x53    - Command (Silence)
0x00    - Error (success)
```

Halt Command

Takes no arguments. Issuing the Halt command will cause the E2ES to enter Idle mode. Data will not be logged and the Radio interface will be down.

Ex:

```
TX ->
0x01    - Endian
0x48    - Command (Halt)
```

```
RX <-
0x48    - Command (Halt)
0x00    - Error (success)
```

Quell Command

Takes two uint16 type arguments. The first argument represents the Logging Interval in seconds and the second represents the Log Delay in seconds. If the Log Interval is not supplied (0x0000) The E2ES will use its internal default value of 600 seconds. Log Delay functionality is optional and accepts a value of 0. Quelling the E2ES causes all data to be cleared and starts device logging with an optionally new Log Interval.

Ex:

TX ->
0x01 - Endian
0x51 - Command (Quell)
0x01 - Log Interval (300 seconds)
0x2C
0x00 - Log Delay (0 seconds)
0x00

RX <-
0x51 - Command (Quell)
0x00 - Error (success)

Parsing Temperature Data

The individual temperature points are shifted so that their resolution fits in their given bit size for an ideal range. With the current logging scheme, 10 bit values are ‘normalized’ via the following formula:
 $(temp - 500) / 10.0$

For example, in the Temperature command we received a value of 0x028E which is decimal value 654:
 $(654 - 500) / 10.0 = 15.4 \text{ Degrees C}$

Logged data is parsed in the same manner but must first be extracted from its 32 bit Word. Every Block will be a byte count that is equally divisible by 4. This is because the data is compacted into 4 byte (32 bit) Words:



Parsing Mark events is easy; just line up the count to the temperature:

- 0 = No Mark
- 1 = Mark between Start and Temp 1
- 2 = Mark between Temp 1 and Temp 2
- 3 = Mark between Temp 2 and Temp 3

Temperatures are parsed in the same manner as before, the only trick here is removing the 10 bit values from the 32 bit word. This is accomplished with a simple bit shift.

As an example, let’s say we have a Block starting with Word 0xA8BA2285:

$0xA8BA2285 \gg 30 = 2 \Rightarrow$ There was a Mark Event between Temp 1 and Temp 2.
 $(0xA8BA2285 \gg 20) \& 0x3FF = 651 \Rightarrow 15.1 \text{ Degrees C}$
 $(0xA8BA2285 \gg 10) \& 0x3FF = 648 \Rightarrow 14.8 \text{ Degrees C}$
 $(0xA8BA2285) \& 0x3FF = 645 \Rightarrow 14.5 \text{ Degrees C}$

This procedure is repeated for every Word in every Block, sequentially, until all data has been converted.

E2ES Common Operation

The general use case for E2ES is to find it, read some or all of the data available, and then optionally Quell it and/or silence it. If going into storage, Halting is also an option. Since the data storage method and block size may change with future firmware or new hardware it is best to write code that is dynamic based on information from the Info command.

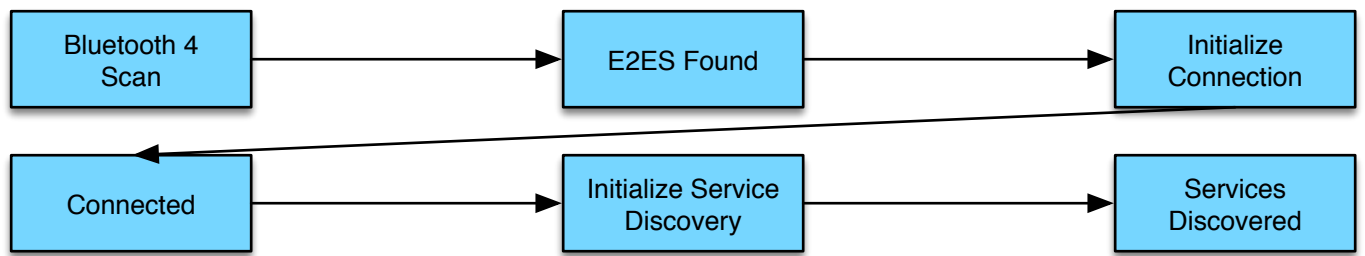
Reading all E2ES data can be accomplished simply by taking the 'Data Points Logged' value and dividing by the 'Data Points per Block' and reading the necessary number of blocks via the Read command.

Reading partial data can be accomplished in a similar fashion as before but only Read the last blocks totaling the number of points you would like to read.

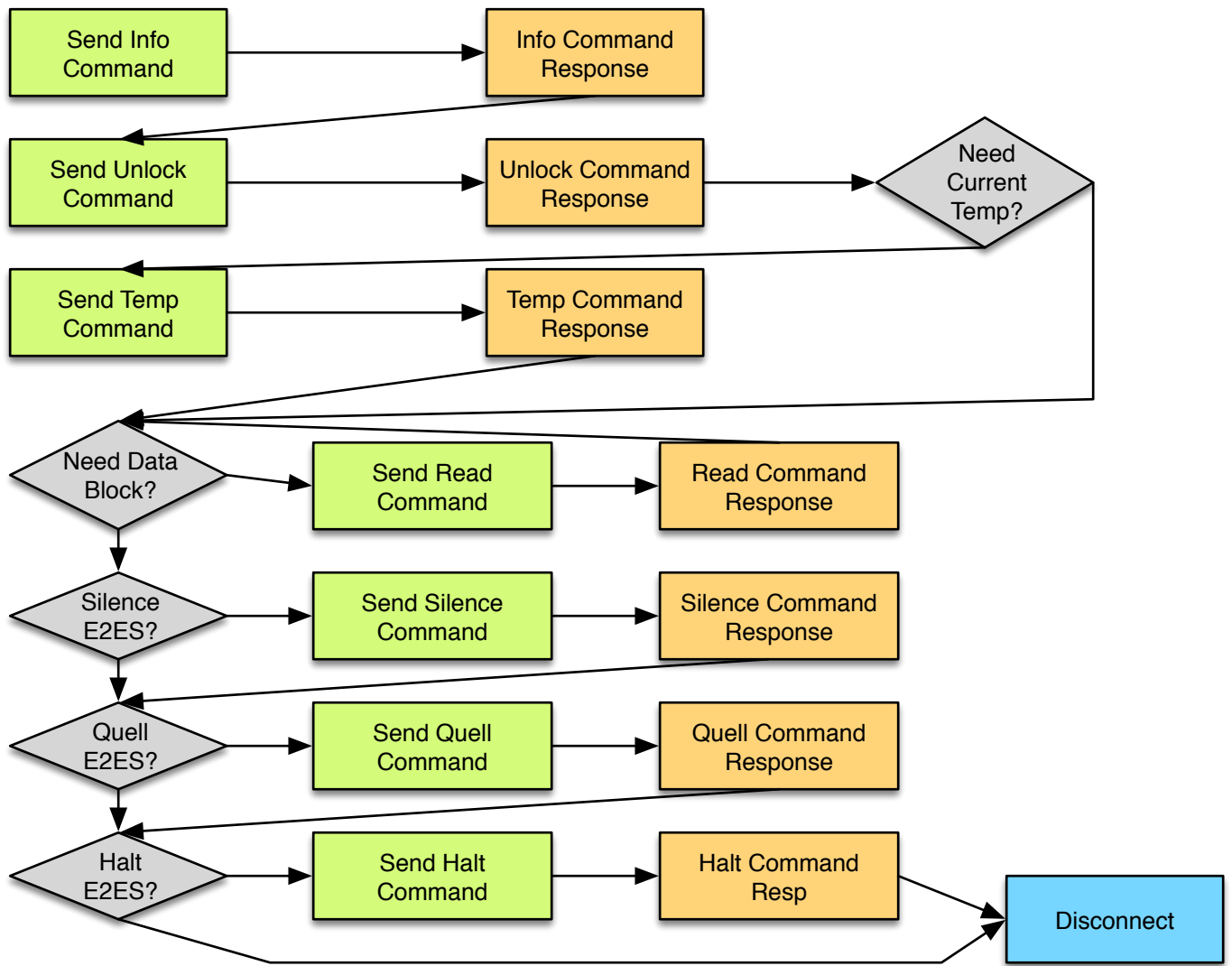
Quick Start



Bluetooth 4.0 General Procedure



Read Procedure



Known Issues

Planned Future Changes

- All Arguments that have values in time (seconds) will be increased to 32 bit types.
- Security Challenge will be calculated and enforced.
- Info command will return seconds since last data point was logged for increased time accuracy.
- Data will be stored in non-volatile memory so it is not lost on error.
- Light Sensor will be added with similar functionality to the button Mark.