**WearDuino – Getting Started**

**Beta Test (WearDuino V005, Document Rev 0.1)**

Thank you for agreeing to be a beta tester for the WearDuino project!

# What Is WearDuino?

WearDuino is a wearable wireless sensor, whose design and firmware is open source and can be changed using the Arduino language. That's why it's called WearDuino! The goal of WearDuino is allowing people to learn about and experiment with wearable wireless sensors, while leaving all data collected in your own hands.

The wireless communication is done using Bluetooth Low Energy (also known as BLE, Bluetooth Smart, Bluetooth 4.0, etc). Most newer smartphones and tablets support BLE, as do most recent laptops and hybrids. Even older computers can support BLE by adding a suitable USB Bluetooth dongle. This means the signals from the WearDuino can be received and processed by any of these devices with suitable software.

WearDuino has a powerful built-in sensor for motion and orientation, that measures acceleration, rotation, and magnetic field. Because each of these three properties is measured is 3 axes (X, Y, Z), it is sometimes called a 9-degree-of-freedom (9DOF) sensor. WearDuino also has a few extra input/output "pins" like an Arduino. These can be used to light up LEDs, monitor switches or controls, or connect additional sensors.

For power, WearDuino uses the simplest possible solution: a single coin-cell battery (CR2032) that is common in watches and many other small devices. The battery life will depend on how many measurements are being made and transmitted. Sending acceleration data every few seconds, the battery should last for months. Under intensive use, continously sending accelerometer/gyro/mag data many times a second, the battery may last for about 25 – 50 hours. There are methods for measuring power consumption which I'll share at a later date.

# Let's Get Started!



If it was not included with your WearDuino, obtain a CR2032 coin cell battery. Orient it so the **grooved edge is** **toward** the circuit board; this means the completely **flat side of the battery (marked +)** **is away** from the circuit board. Slide it fully into the battery retainer.



Unlike stylish, sealed-for-life gadgets, WearDuino has a good old-fashioned on-off switch. It's easier than pulling out the battery when you want to mess with the circuitry or if your new firmware programming goes haywire. Unless you have lilliputian fingertips, a toothpick or paperclip may help. Slide the switch inward to turn it on.

# Communicating with the WearDuino

**Create a Bluetooth Low Energy Central Device in Software**

The WearDuino acts as a Bluetooth **peripheral** device. To communicate with it, your software must use the Bluetooth capabilities of your platform (Linux, Mac OS-X, Windows, iOS, Android) to act as a Bluetooth **central** device.

If you are using node.js, here is a library that supports BLE on Linux/Mac/Windows.

<https://github.com/sandeepmistry/noble>

For devices without built-in Bluetooth, you'll need a compatible Bluetooth Low Energy dongle (see that site for a list).

**Scanning and Connecting to the WearDuino**

Your software should begin scanning, and during scanning the WearDuino should appear as a Bluetooth device advertising the WearDuino *service* with a special UUID.

Wearduino **Service** UUID: 30dff168-62de-11e4-b116-123b93f75cba

Your software should then *connect* to the WearDuino, and can then discover *characteristics* within the service. The WearDuino service has only three *characteristics*: one for **receiving data** from the WearDuino, and for **sending commands** to the WearDuino, and one to request a **disconnect**.

WearDuino **Receive Data** UUID: 30dff169-62de-11e4-b116-123b93f75cba

WearDuino **Send Command** UUID: 30dff16A-62de-11e4-b116-123b93f75cba

WearDuino **Disconnect** UUID: 30dff16B-62de-11e4-b116-123b93f75cba

As you can see, these UUID's differ by only one character, and are in sequence.

**Receiving Data from the WearDuino**

Once connected, the Receive Data characteristic can be *read* at any time. However, for a stream of sensor readings, your software should *subscribe* to *notifications.* In this mode, an event will be triggered in your software every time new data is received.

Bluetooth Low Energy is designed to be very efficient in its use of power, and one way this is accomplished is by keeping the amount of data transmitted as small as possible. In WearDuino, each data packet is just **20 bytes** long.

Since there is only one " channel" for receiving data from the WearDuino, the first byte of the packet is used to indicate the Message Type. When using the Core Demo firmware in the WearDuino, the following table shows how data from the sensors and pins are packed into the remaining bytes.

|  |  |  |
| --- | --- | --- |
| **Byte position** | **Content** | **Options or Comments** |
| 0 | Message type | 00h=Streaming data |
| 1 | Accelerometer X (LSB) | If accelerometer is active, signed 16-bit data. If not, 00h. |
| 2 | Accelerometer X (MSB) |
| 3 | Accelerometer Y (LSB) | If accelerometer is active, signed 16-bit data. If not, 00h. |
| 4 | Accelerometer Y (MSB) |
| 5 | Accelerometer Z (LSB) | If accelerometer is active, signed 16-bit data. If not, 00h. |
| 6 | Accelerometer Z (MSB) |
| 7 | Gyroscope X (LSB) | If gyroscope is active, signed 16-bit data. If not, 00h. |
| 8 | Gyroscope X (MSB) |
| 9 | Gyroscope Y (LSB) | If gyroscope is active, signed 16-bit data. If not, 00h. |
| 10 | Gyroscope Y (MSB) |
| 11 | Gyroscope Z (LSB) | If gyroscope is active, signed 16-bit data. If not, 00h. |
| 12 | Gyroscope Z (MSB) |
| 13 | Mag X (LSB) or  AnalogRead Pin 2 (LSB) | If magnetometer is active, signed 16-bit data.  If analog input is active for Pin 2, unsigned 10-bit data (0-1024).  Otherwise, 00h. |
| 14 | Mag X (MSB) or  AnalogRead Pin 2 (MSB) |
| 15 | Mag Y (LSB) or  AnalogRead Pin 3 (LSB) | If magnetometer is active, signed 16-bit data.  If analog input is active for Pin 3, unsigned 10-bit data (0-1024).  Otherwise, 00h. |
| 16 | Mag Y (MSB) or  AnalogRead Pin 3 (MSB) |
| 17 | Mag Z (LSB) or  AnalogRead Pin 4 (LSB) | If magnetometer is active, signed 16-bit data.  If analog input is active for Pin 4, unsigned 10-bit data (0-1024).  Otherwise, 00h. |
| 18 | Mag Z (MSB) or  AnalogRead Pin 4 (MSB) |
| 19 | DigitalRead Pins 0,1,2,3,4 | For active digital input pins, bit 0 tracks pin 0, bit 1 tracks pin 1, etc. Bit is set to 0 for inactive digital input pins. |

**Sending Commands to the WearDuino**

Just like for received data, the packet length for sending data to the WearDuino is limited to 20 bytes. And just as for receiving data, the first byte of the sent packet is used to indicate the Message Type.

When using the Core Demo firmware in the WearDuino, the following table shows how commands can control the configuration of the WearDuino.

|  |  |  |
| --- | --- | --- |
| **Byte position** | **Content** | **Options or Comments** |
| 0 | Command type | 00h=Configure the Wearduino |
| 1 | Mode selection | 00h=off; 01h=streaming; 02h=logging; 03h=both |
| 2 | Sampling interval | 01h=100 mSec … FFh=2550 mSec;  Default 0Fh is 150 mSec |
| 3 | Accelerometer control | 00h=off; 01h=on during meas; 02h=on when conn; 03h=on always |
| 4 | Gyroscope control | 00h=off; 01h=on during meas; 02h=on when conn; 03h=on always |
| 5 | Magnetometer control | 00h=off; 01h=on during meas; 02h=on when conn; 03h=on always |
| 6 | Pin 0 control | 02h=digital input;  03h=digital output, low;  07h=digital output, high when connected;  0Bh digital output, high during measurement;  0Fh=digital output, high during both |
| 7 | Pin 1 control |
| 8 | Pin 2 control | 00h=analog input (Mag must be inactive);  02h=digital input;  03h=digital output, low;  N5h=analog output (N=high byte), on when connected;  07h=digital output, high when connected;  N9h=analog output (N=high byte), on during measurement;  0Bh digital output, high during measurement;  NDh=analog output (N=high byte), on during both;  0Fh=digital output, high during both |
| 9 | Pin 3 control |
| 10 | Pin 4 control |
| 11 | future use |  |
| 12 | future use |  |
| 13 | future use |  |
| 14 | future use |  |
| 15 | future use |  |
| 16 | future use |  |
| 17 | future use |  |
| 18 | future use |  |
| 19 | future use |  |