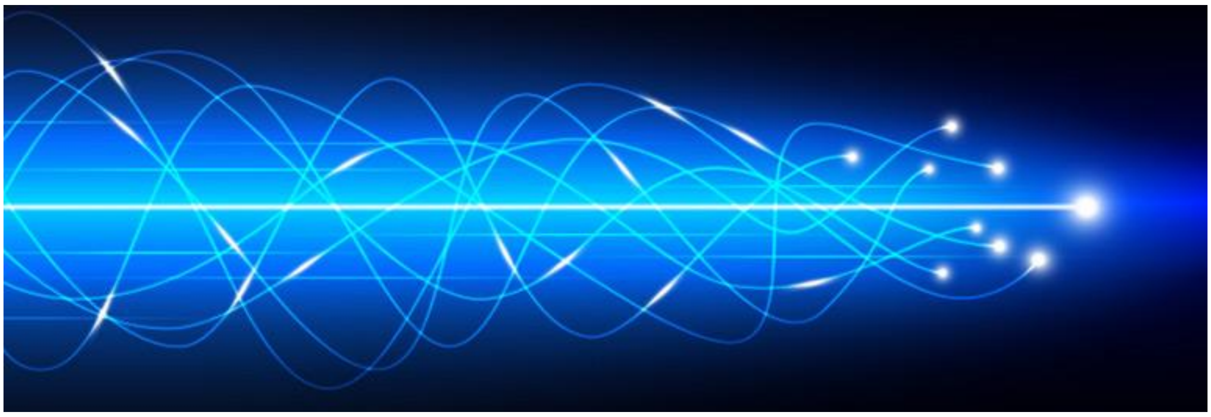
The Web Bluetooth Series

Martin Woolley, Bluetooth SIG

Part 4 - Reading and Writing



Introduction

In the previous part of this series, I covered the use of the Web Bluetooth API to discover Bluetooth Low Energy (LE) devices, connect to a selected device and then discover and cache its GATT services and characteristics.

In this fourth part, we'll learn how to read and write characteristic values.

It is assumed throughout that you have a basic working understanding of Bluetooth LE GATT, including concepts such as services and characteristics.

Project

To complete the hands-on parts of this article, you'll need a Bluetooth LE device which is hosting certain GATT services and characteristics. Review the details I provided in the last part of this series of articles regarding the test LE device you need and other elements of your development environment which are assumed to be in place.

State Validation

Reading and writing characteristic values can only be performed when we have a connection to a device. We also need to know that the connected device contains the required GATT services and characteristics before proceeding with the read or write operation. Therefore service discovery must also have completed. So, at the start of the readModelNumber() and randomLEDs() functions, we will carry out some simple, defensive validation right at the start to check we have achieved the required state. You'll see the code required shortly.

Reading characteristic values

We'll kick the coding off by learning how to read a characteristic value. Your LE test device should contain the GATT device information service and the service should at least contain the Model Name String characteristic. We're going to read its value and display the result in our web page.

Because we took the trouble to track the discovered services and characteristics, we have the opportunity to check that the connected device is suitable for the selected function. This is good practice and helps us give users a better experience.

Update your code so that it includes the following:

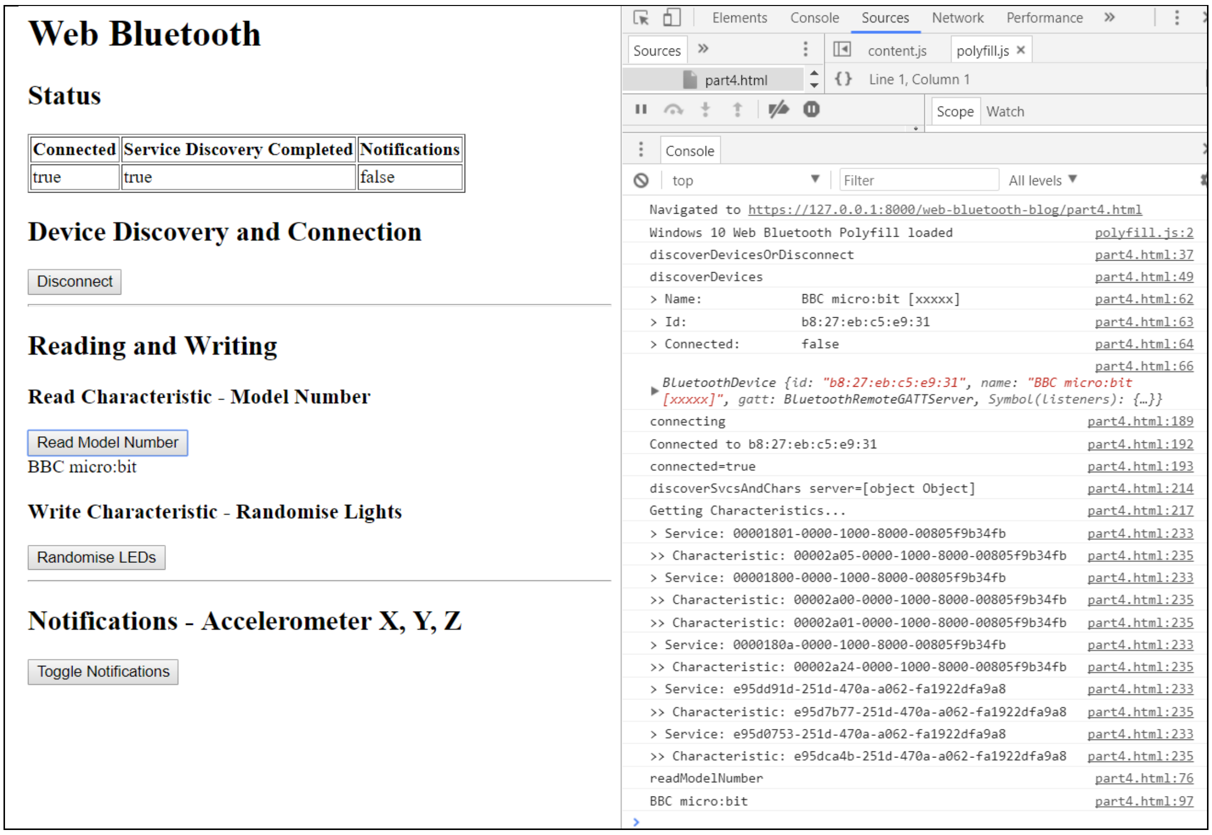
```
function readModelNumber() {
  console.log("readModelNumber");
  // state validation
  if (!connected) {
    alert("Error: Discover and connect to a device before using this function");
    return;
  }
  if (!services_discovered) {
    alert("Error: Service discovery has not yet completed");
    return;
  }
  if (!has_device_information_service) {
    alert("Error: The connected device does not contain the device information
service");
    return;
  }
  if (!has_model_name_string) {
    alert("Error: The connected device does not contain the model name string
characteristic");
    return;
  }
}
```

Test the state validation conditions now. Reload the page in your browser and click the Read Model Number button. Try clicking it before service discovery has completed if you can.

Finally, complete the readModelNumber function to initiate reading the characteristic:

```
function readModelNumber() {
       console.log("readModelNumber");
  // state validation
  if (!connected) {
    alert("Error: Discover and connect to a device before using this function");
    return;
  }
  if (!services_discovered) {
    alert("Error: Service discovery has not yet completed");
    return;
  }
  if (!has_device_information_service) {
    alert("Error: The connected device does not contain the device information
service");
    return;
  }
  if (!has_model_name_string) {
    alert("Error: The connected device does not contain the model name string
characteristic");
    return;
  }
  model_number_string.readValue()
       .then(value => {
              data = new Uint8Array(value.buffer);
              model_number_string = new TextDecoder("utf-8").decode(data);
              console.log(model_number_string);
              document.getElementById("model_number").innerHTML =
model_number_string;
       })
       .catch(error => {
              console.log('Error: ' + error);
              alert('Error: ' + error);
              return;
       });
}
```

Reload index.html in your browser and test. The result should look like this:



Web Bluetooth

Status

Connected	Service Discovery Completed	Notifications
true	true	false

Device Discovery and Connection

Disconnect

Reading and Writing

Read Characteristic - Model Number

Read Model Number

BBC micro:bit

Write Characteristic - Randomise Lights

Randomise LEDs

Notifications - Accelerometer X, Y, Z

Toggle Notifications

```
Navigated to https://127.0.0.1:8000/web-bluetooth-blog/part4.html
Windows 10 Web Bluetooth Polyfill loaded                  polyfill.js:2
discoverDevicesOrDisconnect                               part4.html:37
discoverDevices                                           part4.html:49
> Name:          BBC micro:bit [xxxxx]                    part4.html:62
> Id:            b8:27:eb:c5:e9:31                        part4.html:63
> Connected:     false                                    part4.html:64
                                                          part4.html:66
▶ BluetoothDevice {id: "b8:27:eb:c5:e9:31", name: "BBC micro:bit
  [xxxxx]", gatt: BluetoothRemoteGATTServer, Symbol(listeners): {…}}
connecting                                               part4.html:189
Connected to b8:27:eb:c5:e9:31                           part4.html:192
connected=true                                           part4.html:193
discoverSvcsAndChars server=[object Object]              part4.html:214
Getting Characteristics...                               part4.html:217
> Service: 00001801-0000-1000-8000-00805f9b34fb          part4.html:233
>> Characteristic: 00002a05-0000-1000-8000-00805f9b34fb  part4.html:235
> Service: 00001800-0000-1000-8000-00805f9b34fb          part4.html:233
>> Characteristic: 00002a00-0000-1000-8000-00805f9b34fb  part4.html:235
>> Characteristic: 00002a01-0000-1000-8000-00805f9b34fb  part4.html:235
> Service: 0000180a-0000-1000-8000-00805f9b34fb          part4.html:233
>> Characteristic: 00002a24-0000-1000-8000-00805f9b34fb  part4.html:235
> Service: e95dd91d-251d-470a-a062-fa1922dfa9a8          part4.html:233
>> Characteristic: e95d7b77-251d-470a-a062-fa1922dfa9a8  part4.html:235
> Service: e95d0753-251d-470a-a062-fa1922dfa9a8          part4.html:233
>> Characteristic: e95dca4b-251d-470a-a062-fa1922dfa9a8  part4.html:235
readModelNumber                                           part4.html:76
BBC micro:bit                                             part4.html:97
>
```

Note: if you're using the node.js code provided in Part 3 on an Apple Mac, you won't see the same model number. Instead, you'll see something reflecting the model of Apple computer you're running on.

Writing characteristic values

Next, we'll see what's involved in writing to a characteristic value. Perhaps not surprisingly, the write API is very similar to the read API.

We will write a value to the LED Matrix State characteristic, which on a BBC micro:bit controls which LEDs in its 5x5 LED matrix are switched on and which are switched off. The characteristic has a 5 octet value and the meaning of each octet and the 5 least significant bits of each is as follows:

```
                    Col1 Col2 Col3 Col4 Col5
Octet 0, LED Row 1: bit4 bit3 bit2 bit1 bit0
Octet 1, LED Row 2: bit4 bit3 bit2 bit1 bit0
Octet 2, LED Row 3: bit4 bit3 bit2 bit1 bit0
Octet 3, LED Row 4: bit4 bit3 bit2 bit1 bit0
Octet 4, LED Row 5: bit4 bit3 bit2 bit1 bit0
```

To keep things simple, we'll have our code generate a random value in the range 0 to 31 for each of the 5 octets in the characteristic value. When we write it to a BBC micro:bit, this will cause the LED matrix to display a random pattern each time we click the button on our web page. If you're using a different board, you could log the values received or connect something to your board which you can control with these values. The choice is yours.

Once again, we'll check states and the results of service discovery before we let the write operation proceed.

Update your code as shown:

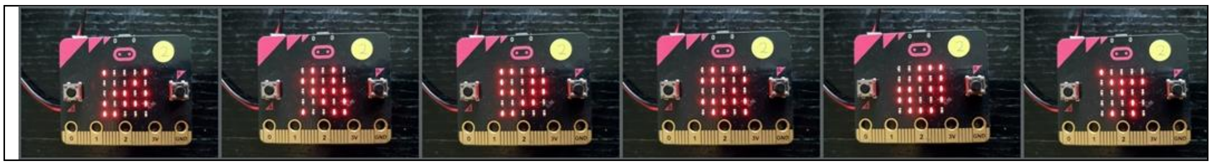
```
function randomLEDs() {
       console.log("randomLEDs");
  // state validation
  if (!connected) {
    alert("Error: Discover and connect to a device before using this function");
    return;
  }
  if (!services_discovered) {
    alert("Error: Service discovery has not yet completed");
    return;
  }
  if (!has_led_service) {
    alert("Error: The connected device does not contain the LED service");
    return;
  }
  if (!has_led_matrix_state) {
     alert("Error: The connected device does not contain the LED matrix state 
characteristic");
     return;
  }
  var led_array = [0, 0, 0, 0, 0];
  led_array[0] = Math.floor(Math.random() * 32);
  led_array[1] = Math.floor(Math.random() * 32);
  led_array[2] = Math.floor(Math.random() * 32);
  led_array[3] = Math.floor(Math.random() * 32);
  led_array[4] = Math.floor(Math.random() * 32);

  var led_matrix_data = new Uint8Array(led_array);

  led_matrix_state.writeValue(led_matrix_data.buffer)
       .then(_ => {
              console.log('LED matrix state changed');
       })
       .catch(error => {
              console.log('Error: ' + error);
              alert('Error: ' + error);
              return;
       });
}
```

Note that the first two validation conditions are the same as the ones we implemented for our read model number function. If you prefer, refactor these two checks into a function you can call from each of the read and write functions.

Reload the index.html page, connect to your test device and click Randomise LEDs several times. If you're testing with a BBC micro:bit, you should see the LED matrix display a sequence of random patterns. Keep an eye on the JavaScript console in your browser too.



Close

You should now know how to read and write to Bluetooth GATT characteristics uding the Web Bluetooth APIs.

In the next and final part we'll learn how to work with Bluetooth notifications.

Call to Action

Web Bluetooth is not yet a W3C standard. I think it needs to be. Web Bluetooth is not yet implemented in all browsers either[1]. And it really needs to be in my humble opinion. Right now, you'll find Web Bluetooth in Chrome on most platforms. The "caniuse" URL in the footnote will give you full information.

If, you feel the same as I do, then I invite and encourage you to petition browser implementers to get behind Web Bluetooth and progress it through the W3C standards process. IoT needs this. You need this.

Mozilla Firefox	https://bugzilla.mozilla.org/show_bug.cgi?id=674737
Microsoft Edge	https://developer.microsoft.com/en-us/microsoft-edge/platform/status/webbluetooth/ https://wpdev.uservoice.com/forums/257854-microsoft-edge-developer/suggestions/9775308-implement-the-web-bluetooth-gatt-client-api
WebKit (Safari)	https://bugs.webkit.org/show_bug.cgi?id=101034

The W3C also host a public email list for Web Bluetooth: https://lists.w3.org/Archives/Public/public-web-bluetooth/

Solution

The solution to the tutorial exercises that we've covered so far is as follows:

```
<html>

<head>
	<script>
		var connected = false;
		var services_discovered = false;
		var selected_device;
		var connected_server;

		// service UUIDs
		ACCELEROMETER_SERVICE = 'e95d0753-251d-470a-a062-fa1922dfa9a8';
		LED_SERVICE = 'e95dd91d-251d-470a-a062-fa1922dfa9a8';
		DEVICE_INFORMATION_SERVICE = '0000180a-0000-1000-8000-00805f9b34fb';

		// presence of services and characteristics
		var has_accelerometer_service = false;
		var has_accelerometer_data = false;

		var has_led_service = false;
```

[1] See https://caniuse.com/#search=web%20bluetooth

```
            var has_led_matrix_state = false;

            var has_device_information_service = false;
            var has_model_name_string = false;

            // characteristic UUIDs
            ACCELEROMETER_DATA = 'e95dca4b-251d-470a-a062-fa1922dfa9a8';
            LED_MATRIX_STATE = 'e95d7b77-251d-470a-a062-fa1922dfa9a8';
            MODEL_NUMBER_STRING = '00002a24-0000-1000-8000-00805f9b34fb';

            // cached characteristics
            var accelerometer_data;
            var led_matrix_state;
            var model_number_string;

            function discoverDevicesOrDisconnect() {
                  console.log("discoverDevicesOrDisconnect: connected=" +
connected);
                  if (!connected) {
                        discoverDevices();
                  } else {
                        selected_device.gatt.disconnect();
                        resetUI();
                  }
            }

            function discoverDevices() {
                  console.log("discoverDevices");

                  var options = {
                        filters: [{ namePrefix: 'BBC' }],
                        optionalServices: [DEVICE_INFORMATION_SERVICE,
ACCELEROMETER_SERVICE, LED_SERVICE]
                  }


                  navigator.bluetooth.requestDevice(options)
                        .then(device => {
                              console.log('> Name:             ' +
device.name);
                              console.log('> Id:               ' + device.id);
                              console.log('> Connected:        ' +
device.gatt.connected);
                              selected_device = device;
                              console.log(selected_device);
                              connect();
                        })
                        .catch(error => {
                              alert('ERROR: ' + error);
                              console.log('ERROR: ' + error);
                        });
            }

            function setConnectedStatus(status) {
                  connected = status;
                  document.getElementById('status_connected').innerHTML =
status;
                  if (status == true) {
                        document.getElementById('btn_scan').innerHTML =
"Disconnect";
                  } else {
                        document.getElementById('btn_scan').innerHTML =
"Discover Devices";
                  }
            }

            function setDiscoveryStatus(status) {
                  services_discovered = status;
```

```
                    document.getElementById('status_discovered').innerHTML =
status;
              }

              function readModelNumber() {
                    console.log("readModelNumber");
                    // state validation
                    if (!connected) {
                          alert("Error: Discover and connect to a device before
using this function");
                          return;
                    }
                    if (!services_discovered) {
                          alert("Error: Service discovery has not yet
completed");
                          return;
                    }
                    if (!has_device_information_service) {
                          alert("Error: The connected device does not contain
the device information service");
                          return;
                    }
                    if (!has_model_name_string) {
                          alert("Error: The connected device does not contain
the model name string characteristic");
                          return;
                    }
                    model_number_string.readValue()
                          .then(value => {
                                data = new Uint8Array(value.buffer);
                                model_number_string = new TextDecoder("utf-
8").decode(data);
                                console.log(model_number_string);

       document.getElementById("model_number").innerHTML = model_number_string;
                          })
                          .catch(error => {
                                console.log('Error: ' + error);
                                alert('Error: ' + error);
                                return;
                          });
              }
              function randomLEDs() {
                    console.log("randomLEDs");
                    // state validation
                    if (!connected) {
                          alert("Error: Discover and connect to a device before
using this function");
                          return;
                    }
                    if (!services_discovered) {
                          alert("Error: Service discovery has not yet
completed");
                          return;
                    }
                    if (!has_led_service) {
                          alert("Error: The connected device does not contain
the LED service");
                          return;
                    }
                    if (!has_led_matrix_state) {
                          alert("Error: The connected device does not contain
the LED matrix state characteristic");
                          return;
                    }
                    var led_array = [0, 0, 0, 0, 0];
                    led_array[0] = Math.floor(Math.random() * 32);
                    led_array[1] = Math.floor(Math.random() * 32);
```

```
                led_array[2] = Math.floor(Math.random() * 32);
                led_array[3] = Math.floor(Math.random() * 32);
                led_array[4] = Math.floor(Math.random() * 32);

                var led_matrix_data = new Uint8Array(led_array);

                led_matrix_state.writeValue(led_matrix_data.buffer)
                    .then(_ => {
                        console.log('LED matrix state changed');
                    })
                    .catch(error => {
                        console.log('Error: ' + error);
                        alert('Error: ' + error);
                        return;
                    });
            }
            function toggleAccelerometerNotifications() {
                console.log("toggleAccelerometerNotifications");
            }

            function connect() {
                if (connected == false) {
                    console.log("connecting");
                    selected_device.gatt.connect().then(
                        function (server) {
                            console.log("Connected to " +
server.device.id);
                            console.log("connected=" +
server.connected);
                            setConnectedStatus(true);
                            connected_server = server;

        selected_device.addEventListener('gattserverdisconnected',
                                onDisconnected);
                            discoverSvcsAndChars();
                        },
                        function (error) {
                            console.log("ERROR: could not connect - "
+ error);
                            alert("ERROR: could not connect - " +
error);
                            setConnectedStatus(false);
                        });
                }
            }

            function onDisconnected() {
                console.log("onDisconnected");
                resetUI();
            }

            function discoverSvcsAndChars() {
                console.log("discoverSvcsAndChars server=" +
connected_server);
                connected_server.getPrimaryServices()
                    .then(services => {
                        has_accelerometer_service = false;
                        has_led_service = false;
                        has_device_information_service = false;

                        services_discovered = 0;
                        service_count = services.length;
                        console.log("Got " + service_count + "
services");

                        services.forEach(service => {
                            if (service.uuid ==
ACCELEROMETER_SERVICE) {
```

```
                                        has_accelerometer_service = true;
                                  }
                                  if (service.uuid == LED_SERVICE) {
                                        has_led_service = true;
                                  }
                                  if (service.uuid ==
DEVICE_INFORMATION_SERVICE) {
                                        has_device_information_service =
true;
                                  }
                                  console.log('Getting Characteristics for
service ' + service.uuid);

      service.getCharacteristics().then(characteristics => {
                                        console.log('> Service: ' +
service.uuid);
                                        services_discovered++;
                                        characteristics_discovered = 0;
                                        characteristic_count =
characteristics.length;

      characteristics.forEach(characteristic => {

      characteristics_discovered++;
                                              console.log('>>
Characteristic: ' + characteristic.uuid);
                                              if (characteristic.uuid ==
ACCELEROMETER_DATA) {
                                                    accelerometer_data =
characteristic;

      has_accelerometer_data = true;
                                              }
                                              if (characteristic.uuid ==
LED_MATRIX_STATE) {
                                                    led_matrix_state =
characteristic;
                                                    has_led_matrix_state
= true;
                                              }
                                              if (characteristic.uuid ==
MODEL_NUMBER_STRING) {
                                                    model_number_string =
characteristic;
                                                    has_model_name_string
= true;
                                              }
                                              if (services_discovered ==
service_count && characteristics_discovered == characteristic_count) {
                                                    console.log("FINISHED
DISCOVERY");

      setDiscoveryStatus(true);
                                              }
                                        });
                                  });
                            });
                      });
            }

            function resetUI() {
                  setConnectedStatus(false);
                  setDiscoveryStatus(false);
                  document.getElementById('model_number').innerHTML = "";
                  document.getElementById('accelerometer_data').innerHTML = "";
            }

      </script>
```

```
</head>

<body>
       <h1>Web Bluetooth</h1>
       <h2>Status</h2>
       <table border="1">
              <tr>
                     <td>
                            <b>Connected</b>
                     </td>
                     <td>
                            <b>Service Discovery Completed</b>
                     </td>
                     <td>
                            <b>Notifications</b>
                     </td>
              </tr>
              <tr>
                     <td id="status_connected">false</td>
                     <td id="status_discovered">false</td>
                     <td id="status_notifications">false</td>
              </tr>
       </table>
       <h2>Device Discovery and Connection</h2>
       <button id="btn_scan" onclick="discoverDevicesOrDisconnect()">Discover
Devices</button>
       <hr>
       <h2>Reading and Writing</h2>
       <h3>Read Characteristic - Model Number</h3>
       <button id="btn_read" onclick="readModelNumber()">Read Model
Number</button>
       <div id="model_number"></div>
       <h3>Write Characteristic - Randomise Lights</h3>
       <button id="btn_write" onclick="randomLEDs()">Randomise LEDs</button>
       <hr>
       <h2>Notifications - Accelerometer X, Y, Z</h2>
       <button id="btn_notify"
onclick="toggleAccelerometerNotifications()">Toggle Notifications</button>
       <div id="accelerometer_data"></div>
</body>

</html>
```