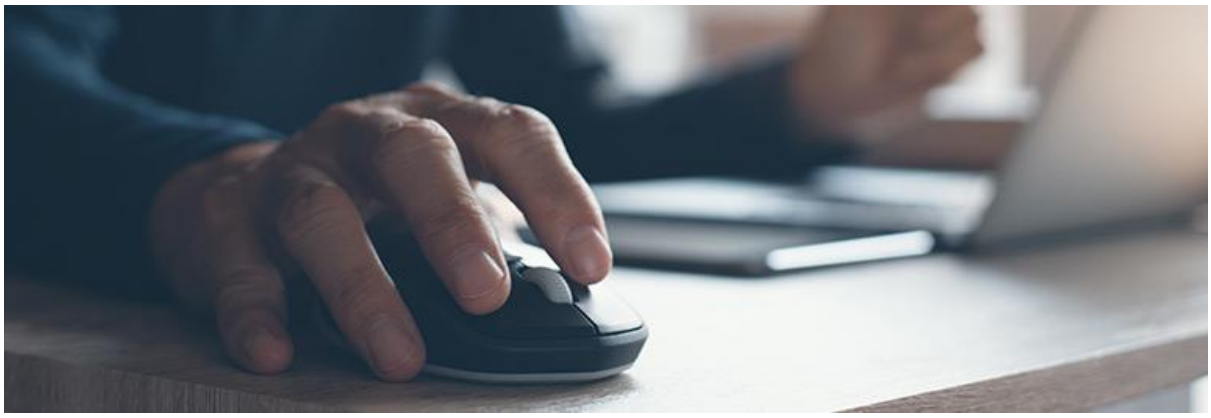
The Web Bluetooth Series

Martin Woolley, Bluetooth SIG

Part 3 - Discovery and Connecting



Introduction

In the first two parts of this series, I explained why I think Web Bluetooth is important and reviewed the foundational, technical knowledge you need to be able to use it.

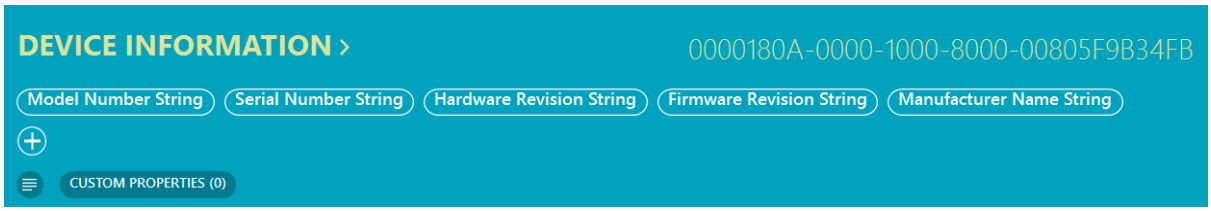
In this third part, we'll start to look at the API itself and how to use it. We'll be focusing on device discovery, connecting to a Bluetooth Low Energy (LE) device and how to perform GATT service discovery.

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

Project

So that you can get some hands-on experience, I'll be using examples that will work with any Bluetooth LE device which implements the right GATT services. The selected services are available for the BBC micro:bit, so if you have a micro:bit already, you just need to install a hex file that contains the required services. If not, you can easily implement what you need on whatever LE developer board you have, including an Arduino or a Raspberry Pi.

For reading values, we'll use the Model Number String characteristic in the device information service:



To learn about writing values, we'll use the LED Matrix State characteristic in the micro:bit LED service which allows us to control what's on the LED display of the device:

For working with notifications, we'll use the micro:bit accelerometer data characteristic in the micro:bit accelerometer service:

A suitable micro:bit hex file for use with this set of tutorials can be created from this MakeCode project.

Full details of the micro:bit services can be found here: https://lancaster-university.github.io/microbit-docs/ble/profile/

If you'd prefer to use another developer board, feel free to implement the required services and characteristics yourself. Source code for the BBC micro:bit is provided at the end of this article. And because I'm nice, I've also provided code for an Arduino 101 and for a Raspberry Pi or similar, running node.js.

Getting Started

You can proceed by reading only or you can join in and do some coding. If you decide on the latter course of action, you'll need to prepare a few things.

Platform

You need to use a web browser which supports Web Bluetooth on a machine which includes a Bluetooth LE stack. Your primary choices of browser are Google Chrome on any of Mac OS, Linux or Android.

You can use Web Bluetooth with Chrome on Windows 10 but you'll need to install a polyfill. See https://github.com/urish/web-bluetooth-polyfill

Web Server

Web Bluetooth security rules stipulate that code must be loaded over HTTPS i.e. from a TLS secured origin. Therefore, you'll need a web server to deploy our code on so that you can access it from your browser. You can use any web server you like, provided it's been set up with a certificate to allow HTTPS to be used.

I use the Node.js "local-web-server" which is available from here: https://github.com/lwsjs/local-web-server . Instructions on setting up local-web-server for HTTPS use are to be found here: https://github.com/lwsjs/local-web-server/wiki/How-to-launch-a-secure-local-web-server-(HTTPS)

Source Code

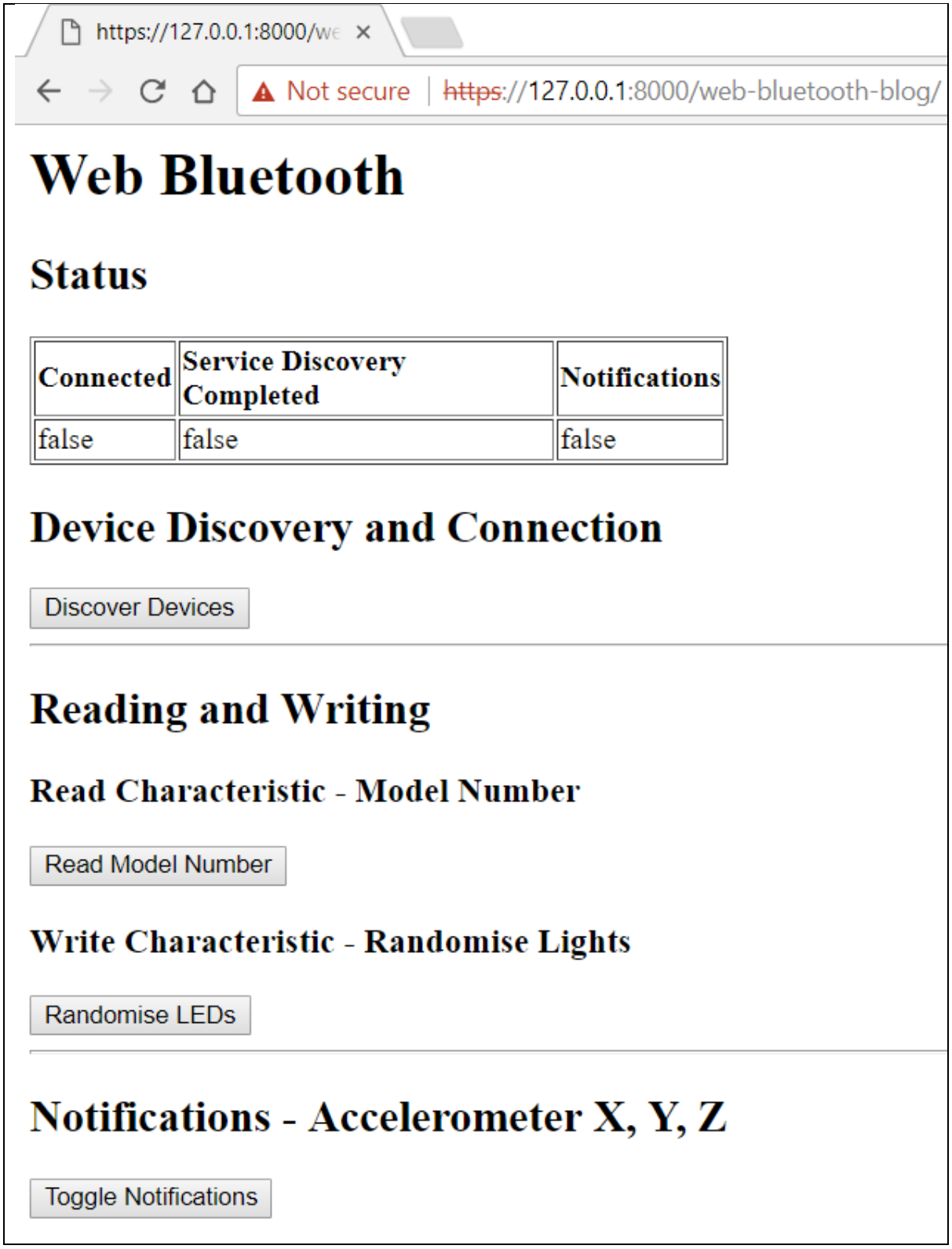
Over the course of this and the next two parts of this series, we'll implement a number of use cases with Web Bluetooth. We'll implement everything in a single HTML file with inline JavaScript and no CSS, just to keep things simple. You wouldn't do it this way in a real application and it's not going to look pretty! But that's not our objective.

Paste the following code into your favourite text editor and save it as "index.html" into a suitable folder within your web server's root directory.

```
<html>
<head>
	<script>
```

```
		function discoverDevicesOrDisconnect() {
			console.log("discoverDevicesOrDisconnect");
		}
		function discoverDevices() {
			console.log("discoverDevices");
		}
		function readModelNumber() {
			console.log("readModelNumber");
		}
		function randomLEDs() {
			console.log("randomLEDs");
		}
		function toggleAccelerometerNotifications() {
			console.log("toggleAccelerometerNotifications");
		}
	</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>
```

Load the page into your browser. It should currently look like this.



My browser reports the site as “Not secure” because I’m using a self-signed certificate for testing purposes, by the way.

Let’s get started on the first set of exercises and meet the Web Bluetooth API!

State Tracking

Our UI will need to allow or restrict various functions depending on whether or not we've selected and connected to a LE device and whether or not GATT service discovery has completed. We'll track these two states in boolean variables. We'll also update the Status section of the UI and the text on the Discover Devices button, depending on the current state.

Let's start by adding some variables for state tracking and a couple of functions we can call to update state variables and to reflect this in the UI. I've highlighted the changes in red:

```
<script>
var connected = false;
var services_discovered = false;

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;
}
```

Device Discovery

Our next job is to complete the discoverDevices () function. The discoverDevicesOrDisconnect() will invoke discoverDevices() in response to the associated UI button being clicked, provided we're not already connected to a device. If we are already connected, it will disconnect the device. We'll come to that case later.

discoverDevices() will scan for advertising Bluetooth peripheral devices and offer them in a list for the user to select from and connect to. Web Bluetooth does not allow you to connect to devices without an explicit request via the user interface, from the user. This is another security feature of the API.

Device discovery is performed by the requestDevice API function. It causes a UI dialogue to pop up and to be populated with suitable LE devices, detected by scanning. It's a "thenable" promise which becomes fulfilled when the user picks one of the listed devices or rejected if anything else happens, such as clicking the cancel button. If fulfilled, a JSON parameter object with properties that describe the selected device is passed to the function inside the "then block".

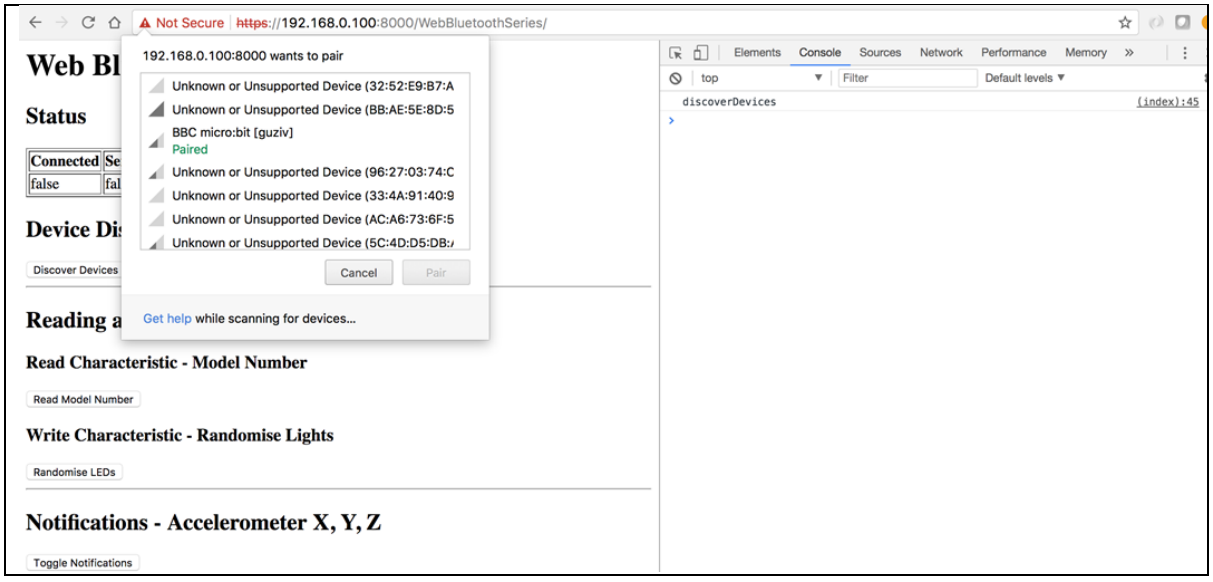
Update your code so that it includes the following:

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

function discoverDevicesOrDisconnect() {
	console.log("discoverDevicesOrDisconnect");
	if (!connected) {
		discoverDevices();
	} else {
```

```
        // TODO disconnect from the current device
        }
}
function discoverDevices() {
        console.log("discoverDevices");

        var options = {
                acceptAllDevices: true
        }

        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);
                })
                .catch(error => {
                        alert('ERROR: ' + error);
                        console.log('ERROR: ' + error);
                });
}
```

Reload the index.html page and open the Chrome developer console so you can watch the log. Now click the Discover Devices button. You should see something like this, with the list content differing according to the LE devices discovered in your environment.

Note: If you are running on Windows and get unexpected results from the Discover Devices button, check the console. If you see this message “Web Bluetooth is experimental on this platform. See https://github.com/WebBluetoothCG/web-bluetooth/blob/gh-pages/implementation-status.md” it means that you have not properly installed and enabled the Web Bluetooth polyfill for Windows, described in the Getting Started section, above.



Select your test device, assuming you see it on the list, and click Pair. Note that in Web Bluetooth “Pair” means “select” rather than the usual meaning in the context of Bluetooth. You’ll see details of the selected device logged to the developer console.

```
discoverDevices
```

```
(index):17 > Name:              BBC micro:bit [guziv]
(index):18 > Id:                kQGGIaP9Qv572mPDwe1cBg==
(index):19 > Connected:         false
(index):23 BluetoothDevice {id: "kQGGIaP9Qv572mPDwe1cBg==", name: "BBC micro:bit
[guziv]", gatt: BluetoothRemoteGATTServer, ongattserverdisconnected: null}
```

Filtering

The way we're using requestDevice currently, all advertising devices are being listed. It's perhaps more usual to filter out devices so that only devices of a type which is relevant to the application, are listed. Filtering is accomplished using the options object which currently looks like this:

```
var options = {
       acceptAllDevices: true
}
```

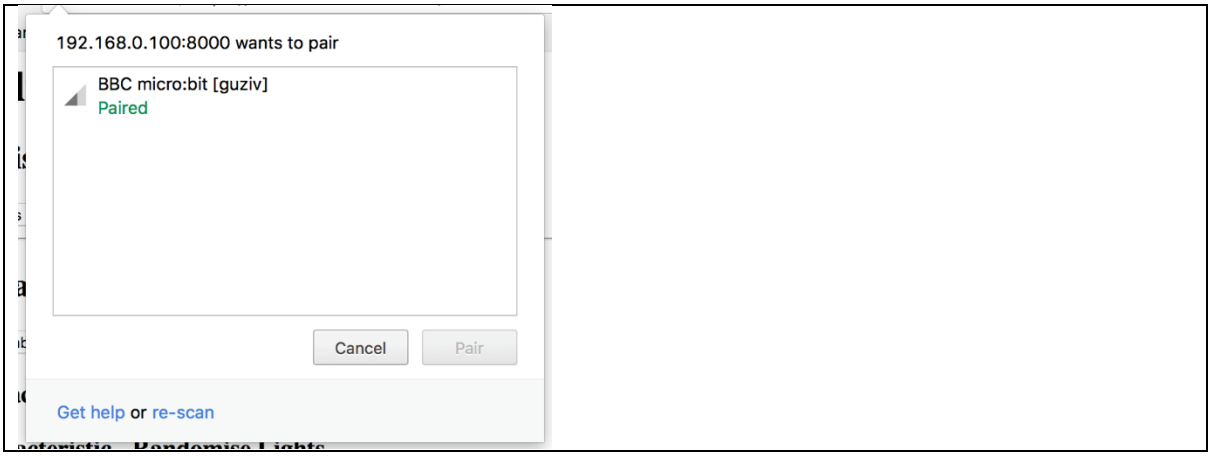
Let's change the options object so that only BBC micro:bits are listed, according to the device name advertised. Furthermore, we'll also indicate the UUIDs of the GATT services we want access to. This is another security requirement.

Note: skip the next bit if you're test device is using the node.js bleno code provided at the end of this article.

Update your code to define the UUIDs of the GATT services we'll be using and to change the options object we use with the requestDevice API as shown here, so that only devices whose advertised name starts with "BBC" are selected.

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

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

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

Reload the index.html page and click Discover Devices. This time you should see only devices whose name starts with "BBC" listed.



Connecting

Next, we'll modify our code so that we connect to the device selected by the user. We're already storing the JSON object representing the selected device. This object includes a BluetoothRemoteGattServer object and it's this object which contains the API function we use to initiate a connection.

```
BluetoothDevice {id: "kQGGIaP9Qv572mPDwe1cBg==", name: "BBC micro:bit [guziv]",
gatt: BluetoothRemoteGATTServer,
ongattserverdisconnected: null}
```

Update the requestDevice code so that a new function, connect() is called:

```
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();
```

and add the following new functions:

```
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);
              },
              function (error) {
                     console.log("ERROR: could not connect - " + error);
                     alert("ERROR: could not connect - " + error);
                     setConnectedStatus(false);
              });
  }
}

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

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

The gatt.connect() call initiates the connection process and returns a BluetoothDevice object in the server parameter when a connection has been established. As you can see, we store the BluetoothDevice object in the connected_server variable so we can use it later.

Reload index.html and test. You'll see something like the following output in the console and if your device has been programmed to visually indicate it has accepted a connection, you'll see it respond accordingly (I always display a "C" on the micro:bit LED display when it accepts a connection).

```
discoverDevices
(index):23 > Name:              BBC micro:bit [guziv]
(index):24 > Id:                kQGGIaP9Qv572mPDwe1cBg==
(index):25 > Connected:         false
(index):27 BluetoothDevice {id: "kQGGIaP9Qv572mPDwe1cBg==", name: "BBC micro:bit
[guziv]", gatt: BluetoothRemoteGATTServer, ongattserverdisconnected: null}
(index):48 connecting
(index):51 Connected to kQGGIaP9Qv572mPDwe1cBg==
(index):52 connected=true
```

Services and Characteristics

Our penultimate task is to discover the GATT services and characteristics which are on the connected device. We need to make a note of the services discovered so that we can validate that the device is suitable for our purposes. We also need to cache those characteristics we want to work with.

Add some variables which we'll use shortly:

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

var has_led_service = false;
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;
```

Now update your code so that when a connection has been established, a call to a new function, discoverSvcsAndChars is made.

```
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();
       },
```

And add the discoverSvcsAndChars function itself:

```
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);
						}
					});
				});
			});
		});
}
```

Our new code works like this; we make a call to the getPrimaryServices function of the BluetoothDevice object which we were given when the connection was established. This produces

an argument called services which contains an array of BluetoothRemoteGATTService objects, which we then iterate through. For each service, we iterate through its array of BluetoothRemoteGATTCharacteristic objects. Boolean variables get set to note which of the services and characteristics we're interested in were discovered and we store the characteristic objects themselves.

Reload and test and your console should look something like this:

```
discoverDevices
(index):39 > Name:             BBC micro:bit [guziv]
(index):40 > Id:               kQGGIaP9Qv572mPDwe1cBg==
(index):41 > Connected:        false
(index):43 BluetoothDevice {id: "kQGGIaP9Qv572mPDwe1cBg==", name: "BBC micro:bit
[guziv]", gatt: BluetoothRemoteGATTServer, ongattserverdisconnected: null}
(index):64 connecting
(index):67 Connected to kQGGIaP9Qv572mPDwe1cBg==
(index):68 connected=true
(index):89 discoverSvcsAndChars server=[object BluetoothRemoteGATTServer]
(index):92 Getting Characteristics...
(index):108 > Service: e95d0753-251d-470a-a062-fa1922dfa9a8
(index):110 >> Characteristic: e95dfb24-251d-470a-a062-fa1922dfa9a8
(index):110 >> Characteristic: e95dca4b-251d-470a-a062-fa1922dfa9a8
(index):108 > Service: e95dd91d-251d-470a-a062-fa1922dfa9a8
(index):110 >> Characteristic: e95d0d2d-251d-470a-a062-fa1922dfa9a8
(index):110 >> Characteristic: e95d93ee-251d-470a-a062-fa1922dfa9a8
(index):110 >> Characteristic: e95d7b77-251d-470a-a062-fa1922dfa9a8
(index):108 > Service: 0000180a-0000-1000-8000-00805f9b34fb
(index):110 >> Characteristic: 00002a26-0000-1000-8000-00805f9b34fb
(index):110 >> Characteristic: 00002a24-0000-1000-8000-00805f9b34fb
FINISHED DISCOVERY
```

The Service Discovery Completed panel in the Status section of the UI should now contain "true".

If you don't get this result, check your code and if necessary, replace with the full solution to this part of the tutorials below. Make sure your LE device does have the three GATT services and child characteristics we're working with and that they have the correct UUIDs.

Disconnecting

Last, we still need to disconnect from the remote device when necessary. Update the discoverDevicesOrDisconnect function to complete the else clause.

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

Close

We've accomplished a lot in this tutorial. You should now have a good feel for how device discovery works in Web Bluetooth, including how to filter unwanted devices and how to ensure the services we will want to access are accessible. You can also create a connection and then discover and cache services and characteristics.

In the next part we'll learn how to read and write characteristic values.

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;
		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';
```

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

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

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

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

			setConnectedStatus(false);
			setDiscoveryStatus(false);

			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 readModelNumber() {
			console.log("readModelNumber");
		}
		function randomLEDs() {
			console.log("randomLEDs");
		}

		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 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 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>
```

Peripheral Device Code

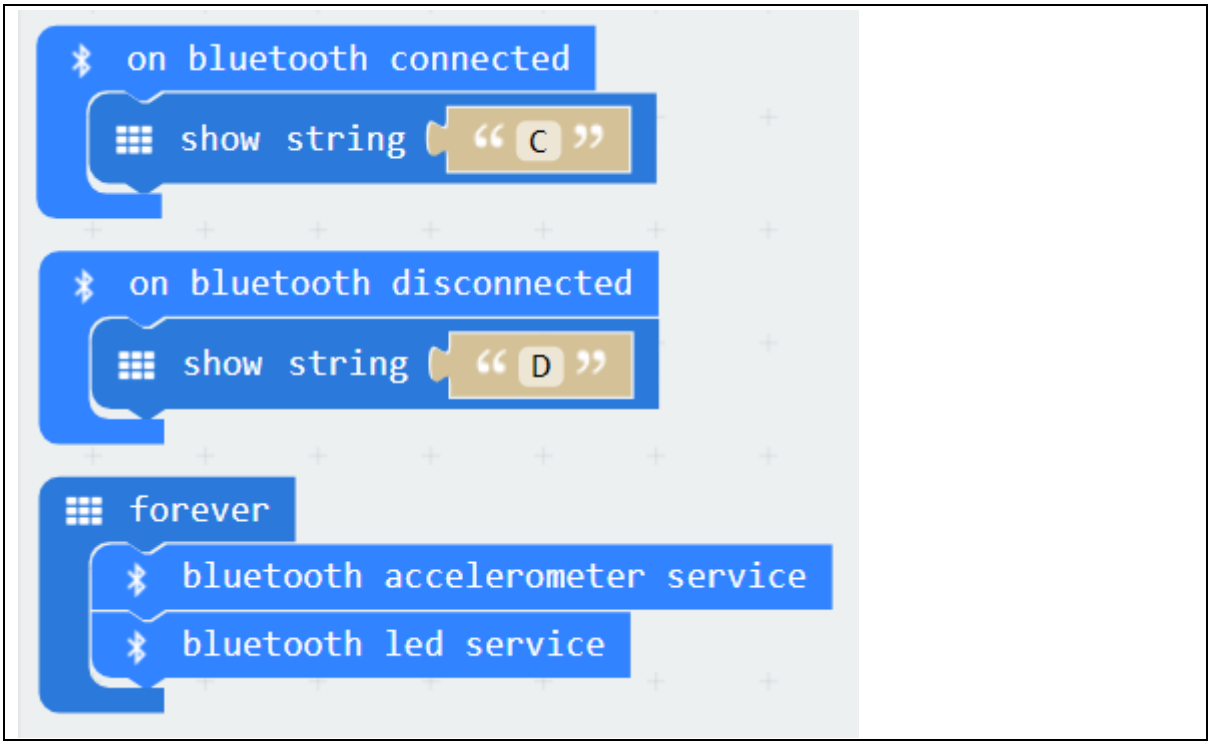
BBC micro:bit - C/C++ using Yotta

```
#include "MicroBit.h"
MicroBit uBit;

void onConnected(MicroBitEvent)
{
    uBit.display.print("C");
}

void onDisconnected(MicroBitEvent)
{
    uBit.display.print("D");
}

int main()
{
    // Initialise the micro:bit runtime.
    uBit.init();

    uBit.display.print("X");

    uBit.messageBus.listen(MICROBIT_ID_BLE, MICROBIT_BLE_EVT_CONNECTED,
onConnected);
    uBit.messageBus.listen(MICROBIT_ID_BLE, MICROBIT_BLE_EVT_DISCONNECTED,
onDisconnected);

    // services: note that the device information service is included by default
    // see config.json property microbit-dal.bluetooth. device_info_service

    new MicroBitAccelerometerService(*uBit.ble, uBit.accelerometer);
    new MicroBitLEDService(*uBit.ble, uBit.display);
```

```
    release_fiber();
}
```

BBC micro:bit - https://makecode.microbit.org/

Edit and run the code live at: https://makecode.microbit.org/_fFH7fodFJ7U3



Arduino 101

```
#include <SoftwareSerial.h>

int connected = 0;

// bluetooth
#include <CurieBLE.h>

// BLE objects
BLEPeripheral blePeripheral;

// GAP properties
char device_name[] = "BBC micro:bit [xxxxx]";

// Characteristic Properties
unsigned char model_number_string_props = BLERead | 0;
unsigned char led_matrix_state_props = BLEWrite | BLERead | 0;
unsigned char accelerometer_data_props = BLERead | BLENotify | 0;

// Services and Characteristics
BLEService device_information_service("180A");
BLECharacteristic model_number_string("2A24", model_number_string_props, "BBC
micro:bit");

BLEService led_service("E95DD91D-251D-470A-A062-FA1922DFA9A8");
BLECharacteristic led_matrix_state("E95D7B77-251D-470A-A062-FA1922DFA9A8",
led_matrix_state_props, 5);

BLEService accelerometer_service("E95D0753-251D-470A-A062-FA1922DFA9A8");
BLECharacteristic accelerometer_data("E95DCA4B-251D-470A-A062-FA1922DFA9A8",
accelerometer_data_props, 6);
```

```
unsigned char initial_led_state[] = { 0x00, 0x00, 0x00, 0x00, 0x00 };
unsigned char initial_accelerometer_data[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

long timestamp = 0;

void setup(void)
{
  Serial.begin(115200);
  Serial.println(F("Arduino setup"));

  randomSeed(analogRead(0));

// set advertising packet content
  blePeripheral.setLocalName(device_name);
  blePeripheral.setDeviceName(device_name);

// add services and characteristics
  blePeripheral.addAttribute(device_information_service);
  blePeripheral.addAttribute(model_number_string);
  blePeripheral.addAttribute(led_service);
  blePeripheral.addAttribute(led_matrix_state);
  blePeripheral.addAttribute(accelerometer_service);
  blePeripheral.addAttribute(accelerometer_data);
  led_matrix_state.setValue(initial_led_state,5);
  accelerometer_data.setValue(initial_accelerometer_data,6);
  Serial.println("attribute table constructed");

// begin advertising
  blePeripheral.begin();
  Serial.println("advertising");

}

void loop()
{

  // listen for BLE peripherals to connect:
  BLECentral central = blePeripheral.central();

  // if a central is connected to peripheral:
  if (central) {
    connected = 1;

    // while the central is still connected to peripheral:
    while (central.connected()) {
       long current_ms = millis();
       // if 500ms have passed notify with faked accelerometer data
       if (current_ms - timestamp >= 500) {
          timestamp = current_ms;
          notifyAccelerometerData();
       }
       if (led_matrix_state.written()) {
        Serial.println("LED matrix state changed:");
        printDec2Bin(led_matrix_state.value()[0]);
        printDec2Bin(led_matrix_state.value()[1]);
        printDec2Bin(led_matrix_state.value()[2]);
        printDec2Bin(led_matrix_state.value()[3]);
        printDec2Bin(led_matrix_state.value()[4]);
       }
    }
  }
  // when the central disconnects, print it out:
  if (connected == 1) {
    connected = 0;
  }
}
```

```
void notifyAccelerometerData() {

  unsigned char accel_data[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
  // random wobble
  unsigned char X_wobble = random(0, 250);
  accel_data[0] = X_wobble;
  unsigned char Y_wobble = random(0, 250);
  accel_data[2] = Y_wobble;
  unsigned char Z_wobble = random(0, 250);
  accel_data[4] = Z_wobble;

  accelerometer_data.setValue(accel_data,6);
}

void printDec2Bin(char value)
{
   int i = 0;
   for(i = 5; i >= 0; i--){
     if((value & (1 << i)) != 0){
       Serial.print("1");
     }else{
       Serial.print("0");
     }
   }
   Serial.println(" ");
}
```

Arduino usage note:

Show the Arduino IDE Serial Monitor to watch values written to the LED Matrix State characteristic.

RaspBerry Pi and other devices that run node.js

Here's the code for a Raspberry Pi or other device which has an LE stack and runs node.js. You need to install the bleno library using

```
npm install bleno
```

and then place this code in a file called main.js

```
var bleno = require('bleno');

bleno.on('stateChange', function (state) {
  console.log('on -> stateChange: ' + state);

  if (state === 'poweredOn') {
    bleno.startAdvertising('BBC micro:bit [xxxxx]');
  } else {
    bleno.stopAdvertising();
  }
});

bleno.on('accept', function (clientAddress) {
  console.log('on -> accept, client: ' + clientAddress);
  bleno.updateRssi();
});

bleno.on('disconnect', function (clientAddress) {
  console.log("Disconnected from address: " + clientAddress);
});

bleno.on('rssiUpdate', function (rssi) {
  console.log('on -> rssiUpdate: ' + rssi);
});
```

```
bleno.on('advertisingStart', function (error) {
  console.log('on -> advertisingStart: ' + (error ? 'error ' + error :
'success'));

  if (!error) {
    bleno.setServices([
      // device information service - do not include if executing on a Mac
      new bleno.PrimaryService({
        uuid: '180a',
        characteristics: [
          // Model Number String
          new bleno.Characteristic({
            value: null,
            uuid: '2a24',
            properties: ['read'],
            onReadRequest: function (offset, callback) {
              console.log("Read request received");
              callback(this.RESULT_SUCCESS, new Buffer("BBC micro:bit"));
            }
          })
        ]
      }),
      // micro:bit LED service
      new bleno.PrimaryService({
        uuid: 'e95dd91d-251d-470a-a062-fa1922dfa9a8',
        characteristics: [
          // LED matrix state
          new bleno.Characteristic({
            value: null,
            uuid: 'e95d7b77-251d-470a-a062-fa1922dfa9a8',
            properties: ['write'],
            onWriteRequest: function (data, offset, withoutResponse, callback) {
              this.value = data;
              console.log('Write request');
              if (data.length == 5) {
                printDec2Bin(data[0]);
                printDec2Bin(data[1]);
                printDec2Bin(data[2]);
                printDec2Bin(data[3]);
                printDec2Bin(data[4]);
              } else {
                console.log("ERROR: invalid characteristic value length");
              }
              callback(this.RESULT_SUCCESS);
            }
          })
        ]
      }),
      // micro:bit accelerometer service
      new bleno.PrimaryService({
        uuid: 'e95d0753-251d-470a-a062-fa1922dfa9a8',
        characteristics: [
          // Accelerometer Data
          new bleno.Characteristic({
            value: null,
            uuid: 'e95dca4b-251d-470a-a062-fa1922dfa9a8',
            properties: ['read', 'notify'],
            onReadRequest: function (offset, callback) {
              console.log("Read request received");
              callback(this.RESULT_SUCCESS, new Buffer("BBC micro:bit"));
            },
            onSubscribe: function (maxValueSize, updateValueCallback) {
              console.log("subscribed to accelerometer notifications");
              this.intervalId = setInterval(function () {
                accel_data = fakeAccelerometerData();
                console.log(accel_data);
                updateValueCallback(accel_data);
```

```
              }, 1000);
            },

            // If the client unsubscribes, we stop broadcasting the message
            onUnsubscribe: function () {
              console.log("unsubscribed from accelerometer notifications");
              clearInterval(this.intervalId);
            }
          })
        ]
      })
    ]);
  }
});

bleno.on('servicesSet', function (error) {
  console.log('on -> servicesSet: ' + (error ? 'error ' + error : 'success'));
});

function printDec2Bin(value) {
  binary = '';
  i = 0;
  for (i = 5; i >= 0; i--) {
    if ((value & (1 << i)) != 0) {
      binary = binary + "1";
    } else {
      binary = binary + "0";
    }
  }
  console.log(binary);
}

function fakeAccelerometerData() {
  accel_data_buffer = new ArrayBuffer(6);
  accel_data = new Uint8Array(accel_data_buffer);
  // random wobble
  X_wobble = Math.floor(Math.random() * 250);
  accel_data[0] = X_wobble;
  Y_wobble = Math.floor(Math.random() * 250);
  accel_data[2] = Y_wobble;
  Z_wobble = Math.floor(Math.random() * 250);
  accel_data[4] = Z_wobble;
  return accel_data;
}
```

Run with the command

```
sudo BLENO_DEVICE_NAME="BBC micro:bit [xxxxx]" node main.js
```

***Note**: Due to limitations in the bleno API, the device name will appear in the Shortened Local Name field rather than the Complete Local Name field and consequently the Web Bluetooth filtering will not select the device. When you get to the Filtering part of the tutorial, therefore retain the acceptAllDevices filter options and do not implement the device name filtering.*

***Note**: If you're running the above node.js script on an Apple Mac, remove the code shown in red. Apple do not allow you to instantiate the device information service and various other standard services with UUIDs issued by the Bluetooth SIG. The service will automatically be there with device model string name set by Apple to reflect the model of computer you're running on. On mine it says "MacBookPro12,1" for example.*

General Note:

The bare minimum services and characteristics have been implemented and, strictly speaking the services do not comply with their specifications, which contain other mandatory characteristics. But for learning and testing purposes, this is fine.