The Web Bluetooth Series

Martin Woolley, Bluetooth SIG

Part 5 - Working with Notifications and Indications

Introduction

In the previous parts of this series, I've reviewed how to use the Web Bluetooth API to discover Bluetooth Low Energy (LE) devices, form connections, discover and cache GATT services and characteristics and read and write characteristic values.

In this final part, we'll learn how to work with notifications and indications.

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

Notifications and indications are handled in exactly the same way in Web Bluetooth, so I'll just be saying "notifications" from now on.

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 part 3 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.

User Interface Status

Our status table has a slot for tracking whether or not we've subscribed to notifications. We'll maintain the right value from the code we'll be writing in this tutorial.

Add a variable which we can use to track the state of notification subscription:

```
var notifications_enabled = false;
```

Add the following function.

```
function setNotificationsStatus(status) {
        notifications_enabled = status;
        document.getElementById('status_notifications').innerHTML = status;
}
```

And call it from resetUI()

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

State Validation

Subscribing to and receiving notifications can only be performed when we have a connection to a device. We also need to have checked that the connected device contains the required GATT service and characteristic before we try to enable notifications on that characteristic. Therefore we'll include some validation in the code we're going to write to subscribe to accelerometer data notifications in much the same way as we did when reading and writing characteristics in the last tutorial part.

Update the toggleAccelerometerNotifications() function:

```
function toggleAccelerometerNotifications() {
        console.log("toggleAccelerometerNotifications");
        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_accelerometer_service) {
                alert("Error: The connected device does not contain the accelerometer service");
                return;
        }
        if (!has_accelerometer_data) {
                alert("Error: The connected device does not contain the accelerometer data
characteristic");
                return;
        }
}
```

Don't forget. per my comment in the previous part of this series, if you prefer, you could refactor the first two state checks into a single validation function you could call from here.

Enabling and Disabling Notifications

In response to the UI button being clicked, we're going to either start notifications or we're going to unsubscribe and stop them, depending on the current subscription state.

Update the toggleAccelerometerNotifications() function:

```
function toggleAccelerometerNotifications() {
        console.log("toggleAccelerometerNotifications");
        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_accelerometer_service) {
                alert("Error: The connected device does not contain the 
accelerometer service");
                return;
        }
        if (!has_accelerometer_data) {
                alert("Error: The connected device does not contain the 
accelerometer data characteristic");
                return;
        }
        if (!notifications_enabled) {
                accelerometer_data.startNotifications().then(_ => {
                        console.log('accelerometer notifications started');
                        setNotificationsStatus(true);
                })
                        .catch(error => {
                                console.log('Error: ' + error);
                                alert('Error: ' + error);
                                return;
                        });
        } else {
                accelerometer_data.stopNotifications()
                        .then(_ => {
                                console.log('accelerometer notifications stopped');
                                setNotificationsStatus(false);
                        })
                        .catch(error => {
                                console.log('Could not stop accelerometer_data 
notifications: ' + error);
                        });
        }
}
```

Notification Events

We still need some code which will process notifications as they're received. We'll also need to do a bit of tidying up when we stop notifications.

Update the toggleAccelerometerNotifications function:

```
        if (!notifications_enabled) {
                accelerometer_data.startNotifications().then(_ => {
                        console.log('accelerometer notifications started');
                        setNotificationsStatus(true);
                        accelerometer_data.addEventListener('characteristicvaluechanged',
                                onAccelerometerData);
                })
                        .catch(error => {
                                console.log('Error: ' + error);
                                alert('Error: ' + error);
                                return;
                        });
        } else {
                accelerometer_data.stopNotifications()
                        .then(_ => {
```

```
                                        console.log('accelerometer notifications stopped');
                                        setNotificationsStatus(false);

          accelerometer_data.removeEventListener('characteristicvaluechanged',
                                                  onAccelerometerData);
                              })
                              .catch(error => {
                                        console.log('Could not stop accelerometer_data notifications: ' +
error);
                              });
          }
```

As you can see, when a notification for a characteristic is received, an event is fired with the characteristic value attached to it. What we've done in this change is register a function to handle this event. We've also removed the event listener when we unsubscribe.

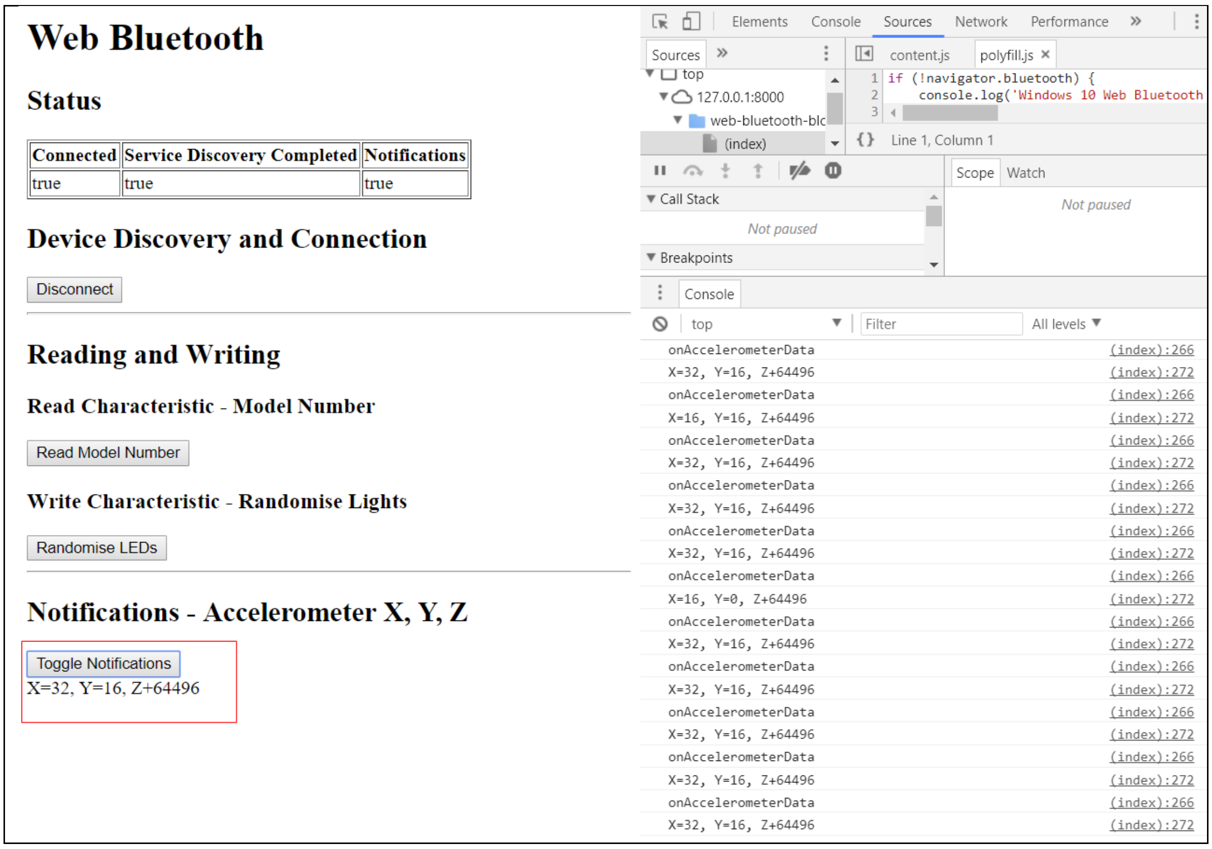
All that's left now is actually process the notification.

Processing Notifications

Our last task is to add the onAccelerometerData function which will be called every time a notification is received. The event handler receives an event object and the characteristic value that we're interested in is sitting inside an ArrayBuffer attached to the value of the event object's target. It's easier to express that last bit in code! Add the following function:

```
function onAccelerometerData(event) {
          console.log("onAccelerometerData");
          buffer = event.target.value.buffer;
          dataview = new DataView(buffer);
          X = dataview.getUint16(0, true);
          Y = dataview.getUint16(2, true);
          Z = dataview.getUint16(4, true);
          console.log("X=" + X + ", Y=" + Y + ", Z+" + Z);
          document.getElementById("accelerometer_data").innerHTML = "X=" + X + ", Y=" + Y + ",
Z=" + Z;
}
```

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



Web Bluetooth

Status

Connected	Service Discovery Completed	Notifications
true	true	true

Device Discovery and Connection

Reading and Writing

Read Characteristic - Model Number

Write Characteristic - Randomise Lights

Notifications - Accelerometer X, Y, Z

X=32, Y=16, Z+64496

```
if (!navigator.bluetooth) {
    console.log('Windows 10 Web Bluetooth
```

```
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=16, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=16, Y=0, Z+64496         (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
onAccelerometerData        (index):266
X=32, Y=16, Z+64496        (index):272
```

Close

This was the last article in this series. We've considered the potential significance of Web Bluetooth, especially with respect to enterprise systems, we've learned some of the foundational knowledge needed to use the Web Bluetooth API and we've gained hands on experience performing fundamental Bluetooth LE tasks such as device discovery, service discovery, reading and writing characteristic values and handling notifications.

I think you're all set to go and do some amazing things with Web Bluetooth now!

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/

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

	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 complete solution to the tutorial exercises is as follows:

```
<html>

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

		// 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';

		// 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 setNotificationsStatus(status) {
                        notifications_enabled = status;
                        document.getElementById('status_notifications').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");
                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_accelerometer_service) {
                        alert("Error: The connected device does not contain
the accelerometer service");
                        return;
                }
                if (!has_accelerometer_data) {
                        alert("Error: The connected device does not contain
the accelerometer data characteristic");
                        return;
                }
                if (!notifications_enabled) {
                        accelerometer_data.startNotifications().then(_ => {
                                console.log('accelerometer notifications
started');
                                setNotificationsStatus(true);

        accelerometer_data.addEventListener('characteristicvaluechanged',
                                        onAccelerometerData);
                        })
                                .catch(error => {
                                        console.log('Error: ' + error);
                                        alert('Error: ' + error);
                                        return;
                                });
                } else {
                        accelerometer_data.stopNotifications()
                                .then(_ => {
                                        console.log('accelerometer notifications
stopped');
                                        setNotificationsStatus(false);

        accelerometer_data.removeEventListener('characteristicvaluechanged',
                                                onAccelerometerData);
                                })
                                .catch(error => {
                                        console.log('Could not stop
accelerometer_data notifications: ' + error);
                                });
                }
        }

        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);
                    setNotificationsStatus(false);
                    document.getElementById('model_number').innerHTML = "";
                    document.getElementById('accelerometer_data').innerHTML = "";
             }

             function onAccelerometerData(event) {
                    console.log("onAccelerometerData");
                    buffer = event.target.value.buffer;
                    dataview = new DataView(buffer);
                    X = dataview.getUint16(0, true);
                    Y = dataview.getUint16(2, true);
                    Z = dataview.getUint16(4, true);
                    console.log("X=" + X + ", Y=" + Y + ", Z+" + Z);
                    document.getElementById("accelerometer_data").innerHTML =
"X=" + X + ", Y=" + Y + ", Z=" + Z;
             }

      </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>
```