**What is a promise?**

A Promise is an object representing the eventual completion or failure of an asynchronous operation. A promise may be created using its constructor. However, most people are consumers of already-created promises returned from functions.

Essentially, a promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.

**Eg:-**

old-style function that expects two callbacks, and calls one of them on eventual completion or failure:

function successCallback(result) {

console.log("It succeeded with " + result);

}

function failureCallback(error) {

console.log("It failed with " + error);

}

doSomething(successCallback, failureCallback);

modern functions return a promise you can attach your callbacks to instead:

let promise = doSomething();

promise.then(successCallback, failureCallback);

…or simply:

doSomething().then(successCallback, failureCallback);

**Uses:-**

Unlike old-style passed-in callbacks, a promise comes with some guarantees:

* Callbacks will never be called before the completion of the current run of the JavaScript event loop.
* Callbacks added with .then even after the success or failure of the asynchronous operation, will be called, as above.
* Multiple callbacks may be added by calling .then several times, to be executed independently in insertion order.

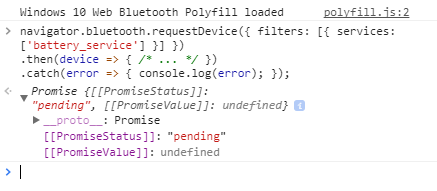
But the most immediate benefit of promises is chaining.

**Promises in Web Bluetooth API:-**

navigator.bluetooth.requestDevice({ filters: [{ services: ['battery\_service'] }] })  
.then(device => { /\* ... \*/ })  
.catch(error => { console.log(error); });

In this requestDevice() will return a device promise which will have device info. Similarly it have many methods like getService(), getCharacteristic() etc which will return promises.

We can check the promiseStatus, promiseValue in console of browser.



**References:-**

<https://developers.google.com/web/fundamentals/primers/promises#promises_arrive_in_javascript>

<https://scotch.io/tutorials/javascript-promises-for-dummies>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises>

**How it works:-**

**User Gesture Required**

As a security feature, discovering Bluetooth devices with **navigator.bluetooth.requestDevice** must be triggered by a user gesture such as a touch or a mouse click.

**Request Bluetooth Devices**

This version of the Web Bluetooth API specification allows websites, running in the Central role, to connect to remote GATT Servers over a BLE connection. It supports communication among devices that implement Bluetooth 4.0 or later.

When a website requests access to nearby devices using **navigator.bluetooth.requestDevice**, Google Chrome will prompt user with a device chooser where they can pick one device or simply cancel the request.

**Image**

[**https://www.google.co.in/search?q=web+bluetooth+api&tbm=isch&source=lnt&tbs=itp:animated&sa=X&ved=0ahUKEwibi8TAkIvXAhUIqI8KHZ4RB8YQpwUIHQ&biw=1366&bih=637&dpr=1#imgrc=ijgZ2\_15nn7x5M**](https://www.google.co.in/search?q=web+bluetooth+api&tbm=isch&source=lnt&tbs=itp:animated&sa=X&ved=0ahUKEwibi8TAkIvXAhUIqI8KHZ4RB8YQpwUIHQ&biw=1366&bih=637&dpr=1#imgrc=ijgZ2_15nn7x5M)**:**

**http://getbootstrap.com/docs/4.0/content/code/**

**requestDevice** function takes a mandatory Object that defines filters. These filters are used to return only devices that match some advertised Bluetooth GATT services and/or the device name.

Example for getting devices advertising **battery\_service**

navigator.bluetooth.requestDevice({ filters: [{ services: ['battery\_service'] }] })  
.then(device => { /\* ... \*/ })  
.catch(error => { console.log(error); });

If your Bluetooth GATT Service is not on the list of the standardized Bluetooth GATT services though, you may provide either the full Bluetooth UUID or a short 16- or 32-bit form.

navigator.bluetooth.requestDevice({  
  filters: [{  
    services: [0x1234, 0x12345678, '99999999-0000-1000-8000-00805f9b34fb']  
  }]  
})  
.then(device => { /\* ... \*/ })  
.catch(error => { console.log(error); });

**Requesting Bluetooth device based on the device name being advertised with the name filters key**

navigator.bluetooth.requestDevice({  
  filters: [{  
    name: 'Francois robot'  
  }],  
  optionalServices: ['battery\_service']  
})

Finally, instead of filters you can use the acceptAllDevices key to show all nearby Bluetooth devices.

navigator.bluetooth.requestDevice({  
  acceptAllDevices: true,  
  optionalServices: ['battery\_service']  
})  
.then(device => { /\* ... \*/ })  
.catch(error => { console.log(error); });

Note: In the above two cases, you will also need to define the optionalServices key to be able to access some services. If you don't, you'll get an error later when trying to access them.

**Connect to a Bluetooth Device**

**requestDevice()** will return a device object through which we can connect to Bluetooth remote GATT Server which holds the service and characteristic definitions. Lets see an example.

navigator.bluetooth.requestDevice({ filters: [{ services: ['battery\_service'] }] })  
.then(device => {  
  // Human-readable name of the device.  
  console.log(device.name);  
  
  // Attempts to connect to remote GATT Server.  
  return device.gatt.connect();  
})  
.then(server => { /\* ... \*/ })  
.catch(error => { console.log(error); });

**Read a Bluetooth Characteristic**

After connecting to GATT service of device then we will get its service and corresponding characteristic as below.

navigator.bluetooth.requestDevice({ filters: [{ services: ['battery\_service'] }] })  
.then(device => device.gatt.connect())  
.then(server => {  
  // Getting Battery Service...  
  return server.getPrimaryService('battery\_service');  
})  
.then(service => {  
  // Getting Battery Level Characteristic...  
  return service.getCharacteristic('battery\_level');  
})  
.then(characteristic => {  
  // Reading Battery Level...  
  return characteristic.readValue();  
})  
.then(value => {  
  console.log('Battery percentage is ' + value.getUint8(0));  
})  
.catch(error => { console.log(error); });

Note that you can also add a characteristicvaluechanged event listener on a characteristic to handle reading its value. Check out [Read Characteristic Value Changed Sample](https://googlechrome.github.io/samples/web-bluetooth/read-characteristic-value-changed.html) to see how to optionally handle upcoming GATT notifications as well.

**Write to a Bluetooth Characteristic**

Writing to a Bluetooth GATT Characteristic is as easy as reading it. This time, let's use the Heart Rate Control Point to reset the value of the Energy Expended field to 0 on a heart rate monitor device.

navigator.bluetooth.requestDevice({ filters: [{ services: ['heart\_rate'] }] })  
.then(device => device.gatt.connect())  
.then(server => server.getPrimaryService('heart\_rate'))  
.then(service => service.getCharacteristic('heart\_rate\_control\_point'))  
.then(characteristic => {  
  // Writing 1 is the signal to reset energy expended.  
  var resetEnergyExpended = Uint8Array.of(1);  
  return characteristic.writeValue(resetEnergyExpended);  
})  
.then(\_ => {  
  console.log('Energy expended has been reset.');  
})  
.catch(error => { console.log(error); });

**Receive GATT Notifications**

Now, let's see how to be notified when the Heart Rate Measurement characteristic changes on the device

navigator.bluetooth.requestDevice({ filters: [{ services: ['heart\_rate'] }] })  
.then(device => device.gatt.connect())  
.then(server => server.getPrimaryService('heart\_rate'))  
.then(service => service.getCharacteristic('heart\_rate\_measurement'))  
.then(characteristic => characteristic.startNotifications())  
.then(characteristic => {  
  characteristic.addEventListener('characteristicvaluechanged',  
                                  handleCharacteristicValueChanged);  
  console.log('Notifications have been started.');  
})  
.catch(error => { console.log(error); });  
  
function handleCharacteristicValueChanged(event) {  
  var value = event.target.value;  
  console.log('Received ' + value);  
  // TODO: Parse Heart Rate Measurement value.  
  // See https://github.com/WebBluetoothCG/demos/blob/gh-pages/heart-rate-sensor/heartRateSensor.js  
}

**References:-**

[**https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web#libraries**](https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web#libraries)

[**https://dev.opera.com/articles/web-bluetooth-intro/**](https://dev.opera.com/articles/web-bluetooth-intro/)

[**https://webbluetoothcg.github.io/web-bluetooth/**](https://webbluetoothcg.github.io/web-bluetooth/)

[**https://www.w3.org/community/web-bluetooth/**](https://www.w3.org/community/web-bluetooth/)