LANTRONIX

# Application Note:

## *xPico® 250 + BLE Application Note*


7535 Irvine Center Drive
Suite 100
Irvine, CA 92618 USA

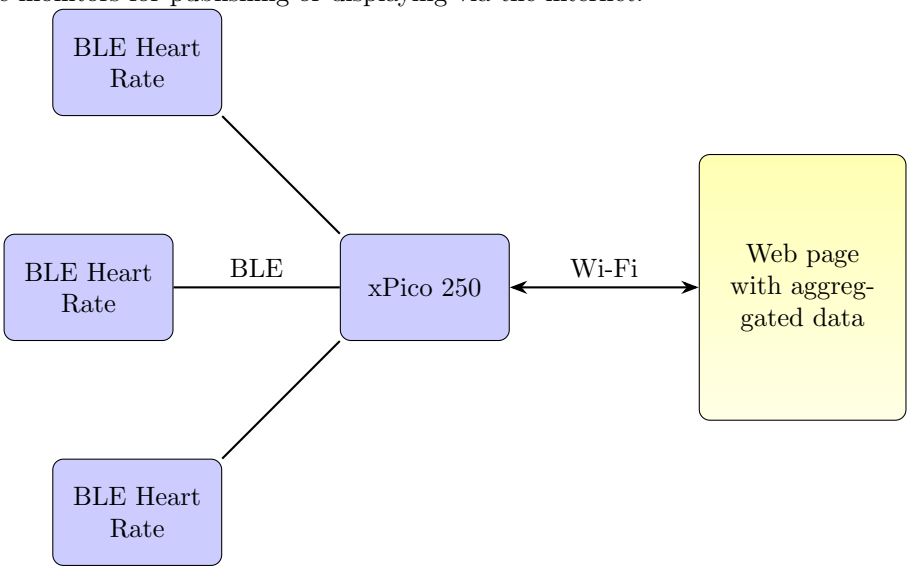
Tel: (800) 526-8766
Tel: +1 (949) 453-3990
Fax: +1 (949) 453-3995
sales@lantronix.com

# Contents

# 1 Introduction

Heart rate sensors are popular devices used for exercise. Many of these sensors use Bluetooth Low Energy (BLE) to connect to another device such as a fitness watch or smartphone for display and tracking of the user's heart rate.

This application note explores how to use an embedded Ethernet, Wi-Fi, and Bluetooth module to create a gateway that aggregates multiple BLE heart rate monitors for publishing or displaying via the internet.



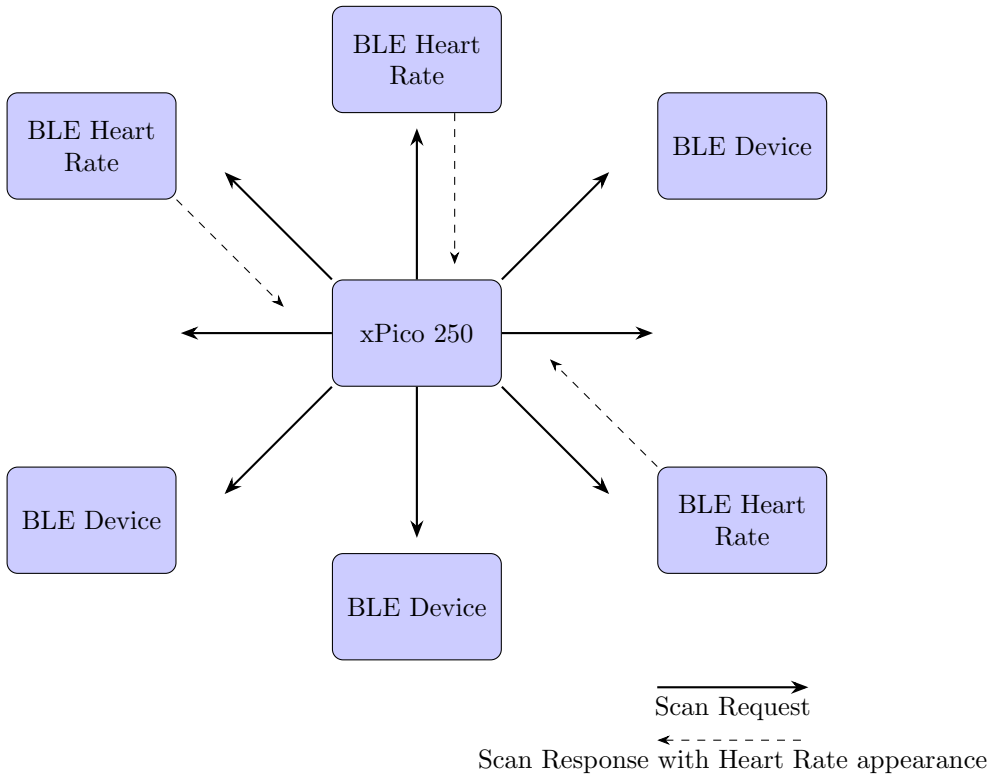

# 2 Generic Access Profile

Every Bluetooth device has the Generic Access Profile (GAP) which contains information such as the class of device, appearance characteristics, and other manufacturer-specific data.[1]

The GAP is the information that is available to other devices without having to initiate a connection first. For devices of the peripheral class, like a Heart Rate Sensor, there are two ways that they will transmit their GAP: advertising and scan response.

The device sends the advertisement at a set interval. The interval time is defined by the device itself. The other option is for a device of type Central to start a scan request. In that case, devices that are not connected will respond with the scan response. For this application, the xPico 250 functions as a central device to scan for peripheral devices.

[1] https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile




Heart rate monitors identify themselves by having a value of 832 (Generic Heart rate Sensor) or 833 (Heart Rate Sensor: Heart Rate Belt) as the appearance in the GAP characteristics. This allows our client to filter which servers to connect to by looking at the scan response data.

To get this data, we first need to issue a scan command. In the call to begin the scan, we pass a callback that is executed when a scan response is received, as well as when the scan process is complete:

```
wiced_bt_ble_scan(BTM_BLE_SCAN_TYPE_HIGH_DUTY, WICED_TRUE,
    ble_scan_callback);
```

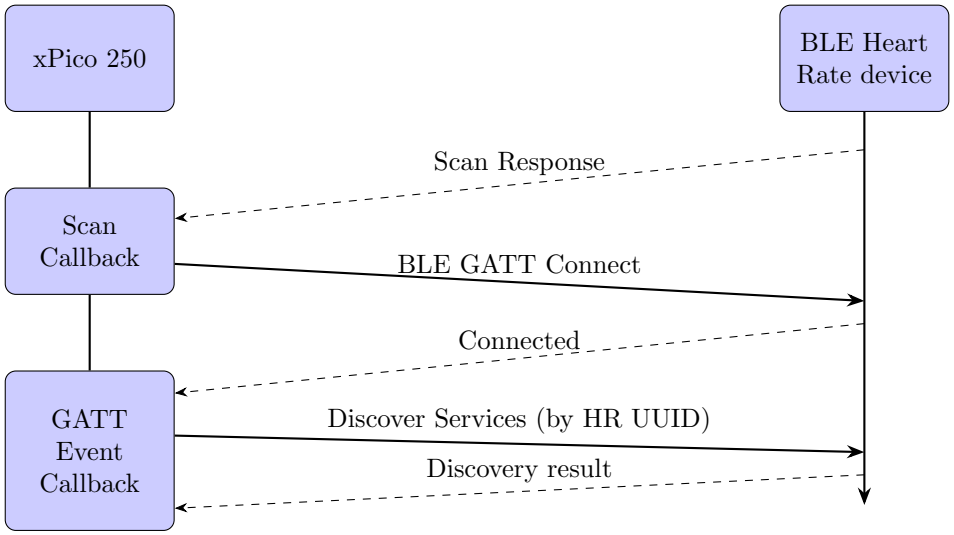
In the ble_scan_callback function, the responses are filtered for the Heart rate sensor appearance values to be placed onto an array that the xPico 250 gateway keeps track of.

For general device design, it is possible to include custom data in the GAP. This would allow a device to broadcast data to multiple clients that will be listening for the advertisement. However this is not how heart rate sensors work, so for this application, the next step is to connect to the device and read the GATT.

# 3 Generic Attributes (GATT)

Once we identify via the GAP that we are interested in a device, we will connect to the device and read the GATT.

## 3.1 Register GATT event callback




Before we can connect to the device in the ble_scan_callback, we need to register another callback to be called when GATT events happen. This is done early in the initialization of the Bluetooth stack:

```
wiced_bt_gatt_register( ble_gatt_event_callback );
```

Where the function prototype for the callback function is:

```
static wiced_bt_gatt_status_t ble_gatt_event_callback(wiced_bt_gatt_evt_t event,
      wiced_bt_gatt_event_data_t *p_event_data);
```

The possible events when this is called are:

**GATT_CONNECTION_STATUS_EVT** The device connects or disconnects

**GATT_DISCOVERY_RESULT_EVT** Discovery results are available

**GATT_DISCOVERY_CPLT_EVT** No more GATT attributes to be discovered

**GATT_OPERATION_CPLT_EVT** A GATT operation, such as a read, write, notify, or indicate

**GATT_ATTRIBUTE_REQUEST_EVT** A GATT attribute request from a remote client. Not applicable in this application since we are the client.

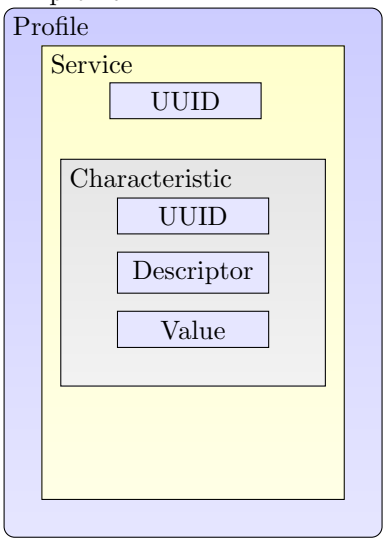
## 3.2 Connect to peripheral

After registering the callback, we can now connect to the device.

```
wiced_bt_gatt_le_connect((message.device)->device_address, (message.device)->addr_type,
        BLE_CONN_MODE_HIGH_DUTY, WICED_TRUE);
```

Once the connection is done the callback with GATT_CONNECTION_STATUS_EVT event is called and we can now read the GATT Profile.

The profile for a device is organized with multiple services. Each service can also have multiple characteristics. The following diagram shows an example GATT profile:




Each profile can include multiple services, and each service can include multiple characteristics.

## 3.3 Discover Heart Rate Service

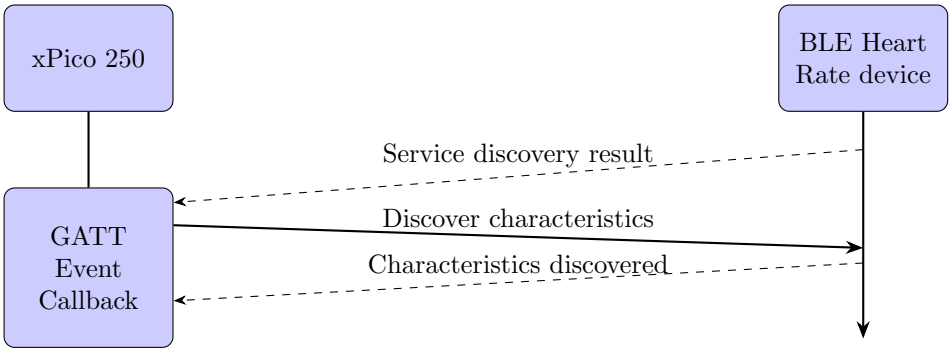
In our application, we discover the address in the GATT (called the handle) that the Heart Rate service is at by discovering by the UUID assigned to the Heart Rate Service[2]:

```
#define HEART_RATE_SERVICE_GATT_UUID        0x180D

memset( &p_discovery_param, 0, sizeof(wiced_bt_gatt_discovery_param_t) );
p_discovery_param.s_handle = 1;
p_discovery_param.e_handle = 0xFFFF;
p_discovery_param.uuid.len = 2;
p_discovery_param.uuid.uu.uuid16 = HEART_RATE_SERVICE_GATT_UUID;
status = wiced_bt_gatt_send_discover(device->conn_id,
```

[2]https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml

```
GATT_DISCOVER_SERVICES_BY_UUID, &p_discovery_param);
```




### 3.4 Discover characteristics

When the service discovery result is received, the GATT event callback is called. We store the handle of the service, and then use that to discover all the characteristics in the service.

```
memset( &p_discovery_param, 0, sizeof(p_discovery_param) );
p_discovery_param.s_handle = service_start_handle;
p_discovery_param.e_handle = service_end_handle;
status = wiced_bt_gatt_send_discover(p_event_data->discovery_complete.conn_id,
      GATT_DISCOVER_CHARACTERISTICS, &p_discovery_param);
```

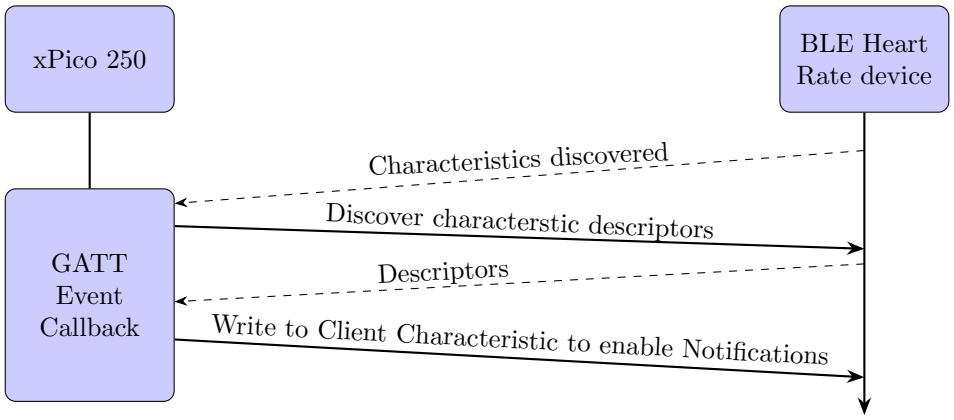
The GATT event callback is called, this time with each characteristic that is inside the service. The Heart Rate Service characteristic has two mandatory characteristics:

- Heart Rate Measurement
- Client Characteristic Configuration

It also has an optional characteristic named Body Sensor Location, and a conditional one named Heart Rate Control Point. We will not be using these two, but for more information you can see the specification[3].

[3] https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml

## 3.5 Discover characteristic descriptors




The Heart Rate Measurement characteristic value cannot be read directly. The specification shows that the value of this characteristic can only be received through notifications. This means that the heart rate sensor device will periodically send a new measurement when it has it.

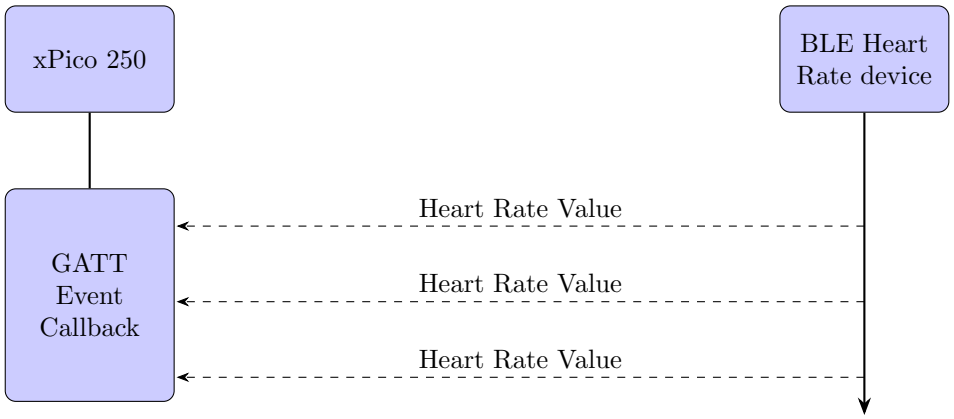
The way to enable the notifications is to write to the Client Characteristic Configuration. So when the GATT event callback is invoked with the characteristic descriptors, the application will now filter by the Client Characteristic Configuration UUID to find the handle.

## 3.6 Enable Notifications

Once all the characteristic descriptors are found, the application writes to the Client Characteristic Configuration to enable notifications:

```
wiced_bt_gatt_value_t write_data;
write_data.auth_req = GATT_AUTH_REQ_NONE;
write_data.handle = temp_handle;
write_data.len = 2;
write_data.value[0] = 0x0001;   // Enable notifications
status = wiced_bt_gatt_send_write(p_event_data->discovery_result.conn_id,
    GATT_WRITE, &write_data);
```

### 3.7 Receiving heart rate value




The device will now begin sending the heart rate value at a periodic interval. Each time a value is received, the GATT event callback is invoked, and the application will extract the value from the notification and store it in memory to be used later when it needs to be displayed on the web page or sent to a cloud system.

## 4 Publish data on local web server

### 4.1 Introduction

The xPico 250 has a built-in webserver that can show custom pages to a device connected to any of the three network interfaces: Ethernet, Wi-Fi Client, and Wi-Fi Soft Access Point. The web server can serve static pages that are stored on the xPico 250's flash file system, and it also allows an SDK application to register a callback on a specific URI to return dynamic data. This application uses both of these capabilities.

### 4.2 Custom URI for dynamic data

The first step is to register a specific URI that a webpage's Javascript code will access to get the dynamic data. This is done by using the built-in http library of the xPico 250 SDK:

```
static const struct ltrx_http_dynamic_callback cb = {
  .uriPath  = "/bleScan",
  .callback = webApiCbFunc
};
void enableRestApi()
{
  ltrx_http_dynamic_callback_register(&cb);
}
```

enableRestApi() is called in the _module_startup() function to register the callback. After that, any HTTP transactions to the /bleScan URI will invoke the webApiCbFunc() function to get the data. This function accesses the array of devices that has been populated with data, and writes it in JSON format as a response.

### 4.3 A simple webpage

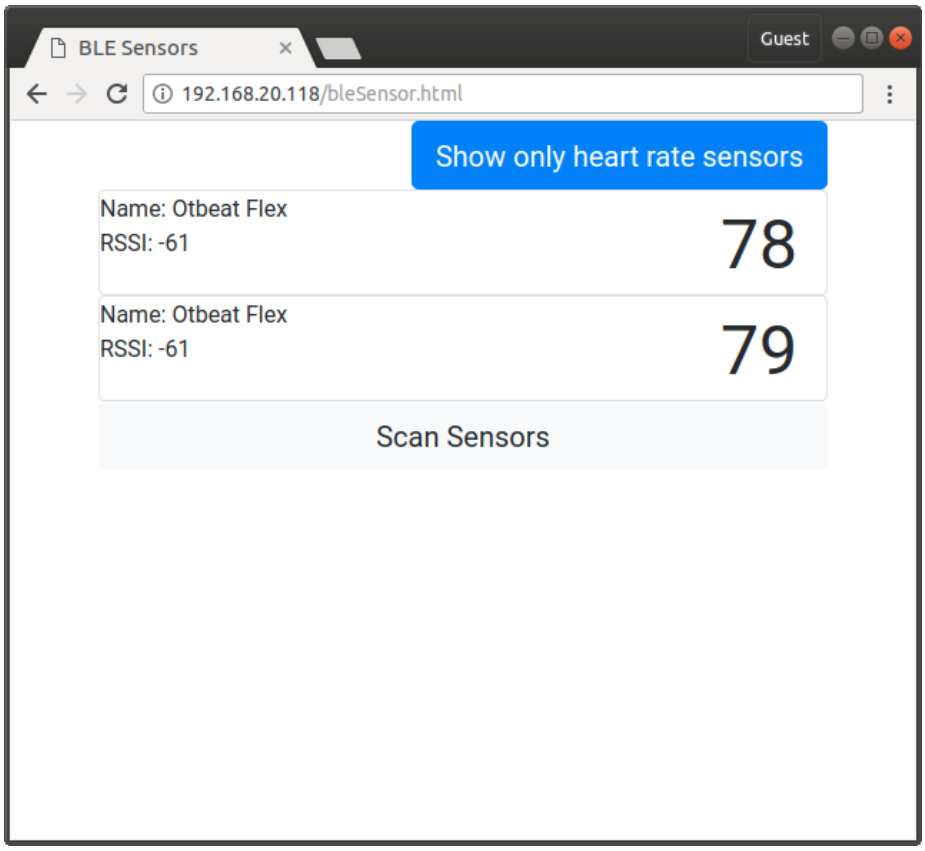
This application includes HTML and Javascript files that are stored in the flash file system of the xPico 250. The HTML page creates a simple block for each heart rate sensor:

```
<div id="templateDisplay" class="deviceBlock border rounded clearfix" style="display:none;">
  <div class="hrDisplay float-right" style="font-size: 48px;margin-right:20px;">0</div>
  <div class="bleName">Name: </div>
  <div class="bleRssi">RSSI: </div>
  <div class="bleAddress">Address: </div>
</div>
```

The Javascript file uses the REST API to gather the data from the application running on the xPico 250 and update the webpage with the results:

```
  $.ajax({
    type: 'GET',
    url: 'bleScan/getScanResults',
    dataType: 'json',
    success: function(response) {
      $("#sensorDisplay").empty();
      $.each(response, function(idx, value) {
        var d = $("#templateDisplay").clone();
        d.prop('id', value.address)
            .appendTo("#sensorDisplay");
        d.find(".bleName").append(value.name);
        d.find(".bleRssi").append(value.rssi);
        d.find(".hrDisplay").text(value.heartrate);
        d.data("appearance",value.appearance);
        d.show();
    });
```

The result is a page that displays the BLE heart rate sensors that are discovered by the application:




# 5 Install and run the application

- Copy the source code from this application into the custom/module directory of the xPico 200 SDK installation. The module should be in a directory called bleSensor.
- Create a project in custom/project
- Ensure that the modules.make file in your project directory includes the bleSensor module
- Build the project
- Update firmware on xPico 250 with the firmware just created
- Create the http directory in the xPico 250 filesystem
- Upload the files from the http directory in the source package to the http directory. Make sure to create the /http/js directory for the appropriate files

- Power on a heart rate monitor
- Access the display with your browser, at: http://ip.address.of.device/bleSensor.html