CONNECTING TO A BLE PRESSURE SENSOR

1. Introduction

The WLB-PT is a battery operated Bluetooth Low Energy (BLE) gauge-pressure-sensor supplied by Hydrotechnik UK. The one used in this application note has a range of −1 to 16 bar; other ranges are available. The sensor broadcasts pressure readings continuously in its BLE advertisement packets, allowing any nearby BLE-capable device, such as a Senquip device, to receive measurements without establishing a connection.

This application note describes how to decode pressure data from the WLB-PT advertisement packet using a Senquip device script. Two implementation methods are described:

- **Method 1** - BLE advertisement scanning is enabled in Senquip device settings. The device collects advertisements and passes them to the script data handler as JSON. The script decodes the pressure value from the JSON.
- **Method 2** - BLE advertisement scanning is disabled in Senquip device settings. The script manages BLE scanning directly using the GAP API, providing finer control and lower overhead when only a single known sensor is being monitored.

The sensor used in this application note advertises under the name WLB-PT.

Figure 1 – WLB-PT Pressure Sensor available from Hydrotechnik UK

2. Bluetooth Low Energy and GAP

Bluetooth Low Energy (BLE) is a wireless communication standard designed for low-power, short-range data transfer. Unlike classic Bluetooth, BLE devices do not need to establish a persistent connection to exchange data. Instead, a device acting as a peripheral, such as the WLB-PT, can continuously broadcast small packets of information to any nearby device that is listening.

2.1. Advertising and Scanning

BLE communication is built around two complementary roles:

- **Advertiser** - a peripheral device that broadcasts data packets at regular intervals. The WLB-PT acts as an advertiser, broadcasting its current pressure reading approximately every second on three dedicated BLE radio channels.
- **Scanner** - a central device that listens for advertisement packets from nearby advertisers. The Senquip device acts as a scanner, receiving and decoding the WLB-PT broadcasts.

Advertisement packets can be received by any scanner within range - typically 10 to 30 metres in an open environment, depending on antenna design and obstructions. No pairing, bonding, or connection handshake is required. The scanner simply listens and decodes.

2.2. GAP — Generic Access Profile

GAP (Generic Access Profile) is the part of the BLE specification that governs how devices advertise themselves and how scanners discover them. It defines the structure of advertisement packets and the procedures used to start and stop scanning.

In the Senquip scripting environment, the GAP API is available after loading api_bt_gap.js. The functions and events relevant to this application are:

Function / Event	Description
GAP.scan(ms, active, filter)	Start a BLE scan. ms sets the scan duration in milliseconds. Active enables active scanning (not required here). Filter limits results to devices whose name starts with the given string.
GAP.EV_SCAN_RESULT	Event fired each time an advertisement packet matching the filter name is received during a scan.
GAP.EV_SCAN_STOP	Event fired when the scan duration expires. Used here to restart the scan immediately for continuous monitoring.
GAP.getScanResultArg(evdata)	Returns the scan result object containing the advertiser address (addr), signal strength (rssi), and raw advertisement payload (advData).
GAP.parseName(advData)	Extracts the complete local name from the advertisement data.

2.3. The Name Filter

The third argument to GAP.scan() is a name prefix filter. When provided, the BLE controller raises EV_SCAN_RESULT events only for advertisers whose device name starts with the filter string. This is important in environments with many BLE devices - without a filter, every advertisement must be evaluated by the script, which can place unnecessary load on the device.

In this application, the filter string **'WLB'** is used. The WLB-PT advertises with the complete local name **'WLB-PT'**, which matches this prefix. Any other BLE devices in range are silently ignored by the BLE controller before they reach the script.

Note: *The name filter is applied in BLE controller hardware, not in the script. This makes filtering very efficient — non-matching advertisements are discarded at the radio level.*

2.4. Advertisement Data

Each EV_SCAN_RESULT event provides a scan result object. The advData field of this object contains the raw advertisement payload as a byte string. The WLB-PT encodes its current pressure reading directly in this payload, so pressure can be obtained from every advertisement packet without establishing a connection to the sensor.

The structure of the WLB-PT advertisement payload is described in the following section.

3. Advertisement Packet Structure

The WLB-PT continuously broadcasts a 22-byte BLE advertisement packet. The pressure value is encoded as a signed 32-bit little endian integer in bytes 10–13, in units of microbar. Dividing by 1,000,000 yields the gauge pressure in bar.

Note: *The MAC address must be entered in the script in lowercase with no colons, e.g. d26dbb0a61ba.*

Byte Offset	Field	Type	Description
0–2	Flags	3 bytes	Standard BLE flags (0x02 0x01 0x06)
3–6	Service Data header	4 bytes	Length, type (0x16), UUID (0x2315)
7	Status / Version	uint8	Always 0x01
8	Status	uint8	0 = normal, 1 = high alarm, 2 = low alarm
9	Reserved	uint8	Always 0x00
10–13	Pressure	int32, little endian	Gauge pressure in microbar. Divide by 1,000,000 for bar.
14–15	Reserved	2 bytes	Always 0x00 0x00
16+	Device name	string	Complete Local Name AD type: 'WLB-PT'

Example packet (at rest, 0 bar gauge):

02 01 06 0b 16 23 15 01 57 00 00 00 00 00 00 00 07 09 57 4c 42 2d 50 54

Example packet (6 bar applied):

02 01 06 0b 16 23 15 01 57 00 9a a2 5e 00 00 00 07 09 57 4c 42 2d 50 54

Bytes 10–13: **9a a2 5e 00** = 0x005EA29A = 6,201,242 microbar ÷ 1,000,000 = **6.20 bar**

4. Method 1 — Settings-Based BLE Scanning

In this method, the Senquip device handles BLE scanning automatically. The device settings are configured to collect BLE advertisement data, which is included in the data handler JSON as a ble array. The script simply parses the JSON and decodes the pressure value.

4.1. Senquip Device Configuration

Enable BLE advertisement scanning in the Senquip device settings. When enabled, the Senquip device scans for BLE advertisement packets from nearby devices and includes them in the data collected and later passed to the data handler in a script.

Optionally, the device id of the sensor is inserted into the *Address Capture List*, which allows the Senquip device to only send data from a specific WLB-PT. The filtering is applied after all Advertising packets have been collected and before transmitting the data to the Portal. Filtering reduces the processing load on the script, but does not reduce effort required by the BLE peripheral.

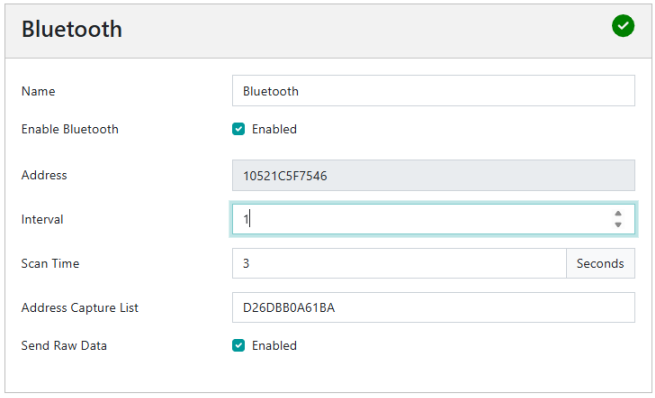


Figure 2 - Bluetooth Configuration for Method 1

Each entry in the ble array contains:

- id — a string containing the device MAC address in lowercase with no colons
- data — a string containing the raw advertisement payload as a lowercase hex string
- rssi — the received signal strength in dBm

Example ble field in the data handler JSON:

```
"ble": [
 {
  "data": "0201060b16231501570000000000000709574c422d5054",
  "rssi": -59,
  "id": "d26dbb0a61ba"
 }
]
```

5. Finding the Sensor MAC Address

The MAC address of the WLB-PT sensor is required to identify it in the ble array. It can be found by:

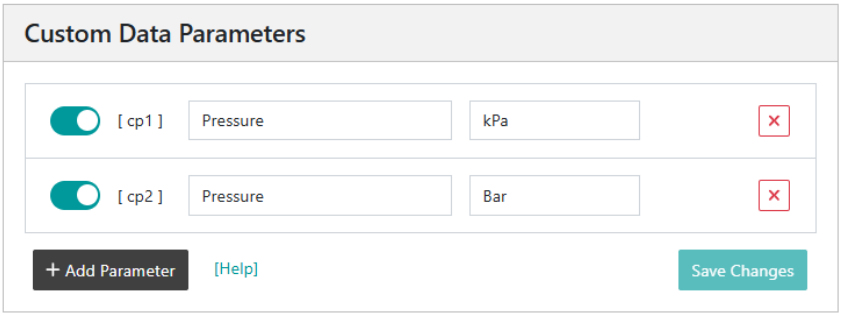
- Using the nRF Connect app (available on Android and iOS) to scan for nearby BLE devices — the WLB-PT will appear as 'WLB-PT'
- Logging the raw JSON from the data handler and identifying the entry whose data field ends with 0709574c422d5054 ('WLB-PT' in hex)

Note: *The MAC address must be entered in the script in lowercase with no colons, e.g. d26dbb0a61ba.*

5.1. Method 1 Script

The data field in the ble array is a hex string. The SQ.parse() function is used to extract the 4-byte little endian pressure value from offset 20 (skipping the first 10 bytes / 20 hex characters). The negative type argument -SQ.S32 reverses the byte order for little endian interpretation.

In both methods, Pressure in kPa will be dispatched to CP1 and pressure in bar will be dispatched to CP2.



Note: *Replace the SENSOR_ID value with the MAC address of your specific sensor.*

```
load('senquip.js');
load('api_endpoint.js');

// MAC address of the WLB-PT sensor (lowercase, no colons)
let SENSOR_ID = 'd26dbb0a61ba';

SQ.set_data_handler(function(data) {
  let d = JSON.parse(data);

  if (!isdef(d.ble)) return;

  for (let i = 0; i < d.ble.length; i++) {
    if (d.ble[i].id === SENSOR_ID) {

      // Bytes 10-13 of advertisement: pressure in microbar
      // hex string offset 20, length 8, little endian signed 32-bit
      let raw = SQ.parse(d.ble[i].data, 20, 8, -16, SQ.S32);
      let pressure_bar = raw / 1000000;
      let pressure_kpa = pressure_bar * 100;

      SQ.dispatch(1, pressure_kpa);   // CP1 = kPa
      SQ.dispatch(2, pressure_bar);   // CP2 = bar
      return;
    }
  }
}, null);
```

6. Method 2 — Script-Based BLE Scanning

In this method, BLE advertisement scanning is disabled in the Senquip device settings. The script manages BLE scanning directly using the GAP API. This approach is more efficient when monitoring a single known sensor, as the name filter passed to GAP.scan() ensures only WLB-PT advertisements are processed at the hardware level.

The script starts a BLE scan on boot and restarts it automatically each time the scan window expires, providing continuous monitoring. When a WLB-PT advertisement is received, the pressure value is decoded directly from the raw byte string.

6.1. Senquip Device Configuration

Leave the Bluetooth peripheral enabled but disable automatic advertisement scanning. All BLE scanning activity will be managed by the script.

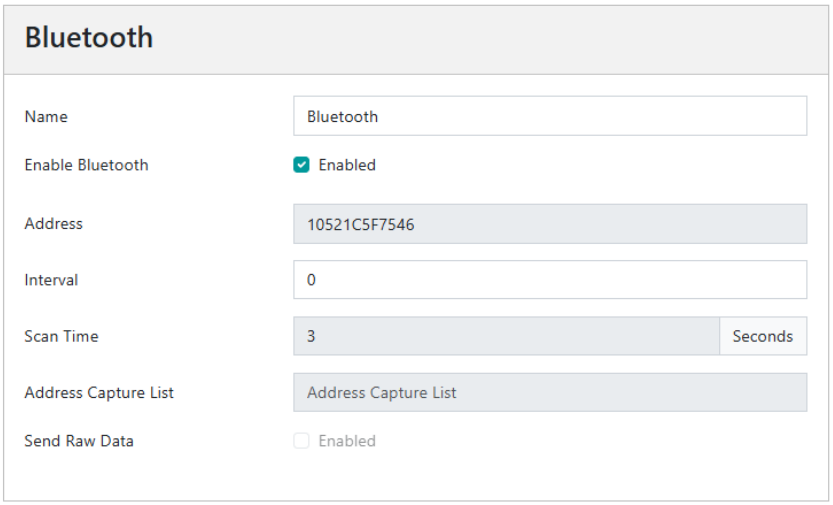


Figure 3 - Bluetooth Peripheral Setup for Method 2

6.2. Method 2 Script

The EV_SCAN_RESULT event is raised each time the BLE controller receives an advertisement from a device matching the 'WLB' name filter. The event handler retrieves the scan result using GAP.getScanResultArg(), which returns an object containing the raw advertisement payload in the advData field as a byte string.

The pressure value is extracted from advData using SQ.parse(). Unlike Method 1 where the advertisement arrives as a hex string in the JSON, advData is a raw binary byte string, so base 0 is used rather than base 16. The four arguments to SQ.parse() work as follows:

- **Offset 10** — skips the first 10 bytes of the packet (flags, service data header, status, and sequence counter), positioning the parser at the start of the pressure field
- **Length 4** — reads 4 bytes, corresponding to the signed 32-bit pressure value
- **Base 0** — instructs SQ.parse() to interpret the input as raw bytes rather than a hex or decimal string
- **-SQ.S32** — the negative sign reverses the byte order before interpreting the result as a signed 32-bit integer, converting from the sensor's little endian format to the host byte order. Without the negative sign, the bytes would be read as big endian and the pressure value would be incorrect.

The raw value is in units of microbar. Dividing by 1,000,000 converts to bar, and multiplying by 100 converts to kPa.

The EV_SCAN_STOP handler restarts the scan immediately when the 5-second scan window expires, ensuring continuous monitoring with no gap between scan windows.

The data handler dispatches the most recently decoded pressure values each time the Senquip device reports to the portal.

The completed script is shown below, and an upgraded version that can listen for multiple pressure sensors is given in Appendix B.

```
load('senquip.js');
load('api_events.js');
load('api_bt_gap.js');
load('api_timer.js');
load('api_endpoint.js');

let SCAN_TIME_MS = 5000;
let pressure_bar = null;
let pressure_kpa = null;

// Optional debug logging - see Debug Logging section
function log(s) {
  UDP.send(s);
}

Event.on(GAP.EV_SCAN_RESULT, function(ev, evdata) {
  let sr = GAP.getScanResultArg(evdata);
  let d = sr.advData;

  // Bytes 10-13: pressure in microbar, little endian signed 32-bit
  // base=0 (raw bytes), -SQ.S32 reverses byte order for little endian
  let raw = SQ.parse(d, 10, 4, 0, -SQ.S32);
  pressure_bar = raw / 1000000;
  pressure_kpa = pressure_bar * 100;

  log('bar=' + JSON.stringify(pressure_bar) +
      ' kPa=' + JSON.stringify(pressure_kpa));
}, null);

Event.on(GAP.EV_SCAN_STOP, function() {
  // Restart scan immediately for continuous monitoring
  GAP.scan(SCAN_TIME_MS, false, 'WLB');
}, null);

SQ.set_data_handler(function() {
  if (pressure_bar !== null) {
    SQ.dispatch(1, pressure_kpa);    // CP1 = kPa
    SQ.dispatch(2, pressure_bar);    // CP2 = bar
  }
}, null);

log('Starting WLB-PT scan...');
GAP.scan(SCAN_TIME_MS, false, 'WLB');
```

6.3. Debug Logging

Method 2 includes an optional log() function that sends debug messages via UDP. This is useful during development and commissioning to verify that the sensor is being found and pressure values are being decoded correctly.

To use debug logging:

1. Run a UDP server on a computer connected to the same network as the Senquip device. The code for a simple Python UDP server is shown in Appendix A.
2. In the Senquip device settings, configure the UDP endpoint with the IP address of the computer running the UDP server.
3. Deploy the script. Log messages will appear on the UDP server as advertisements are received and decoded.

Note: *Remove or comment out all log() calls before deploying to production to avoid unnecessary network traffic.*

7. Conclusion

This application note has demonstrated how to read gauge pressure from the WLB-PT BLE sensor using a Senquip device. By decoding the pressure value directly from the BLE advertisement packet, no connection to the sensor is required - the sensor simply broadcasts its reading continuously and the Senquip device decodes it as part of its normal data collection cycle.

Both implementation methods produce the same result. Method 1 is simpler to implement and is recommended when the Senquip device is already collecting BLE advertisements from multiple sensors. Method 2 is more efficient for dedicated single-sensor deployments, as the hardware-level name filter reduces processing overhead.

Figure 4 below shows the pressure displayed on the Senquip Portal, with the value updating as pressure is applied and released.

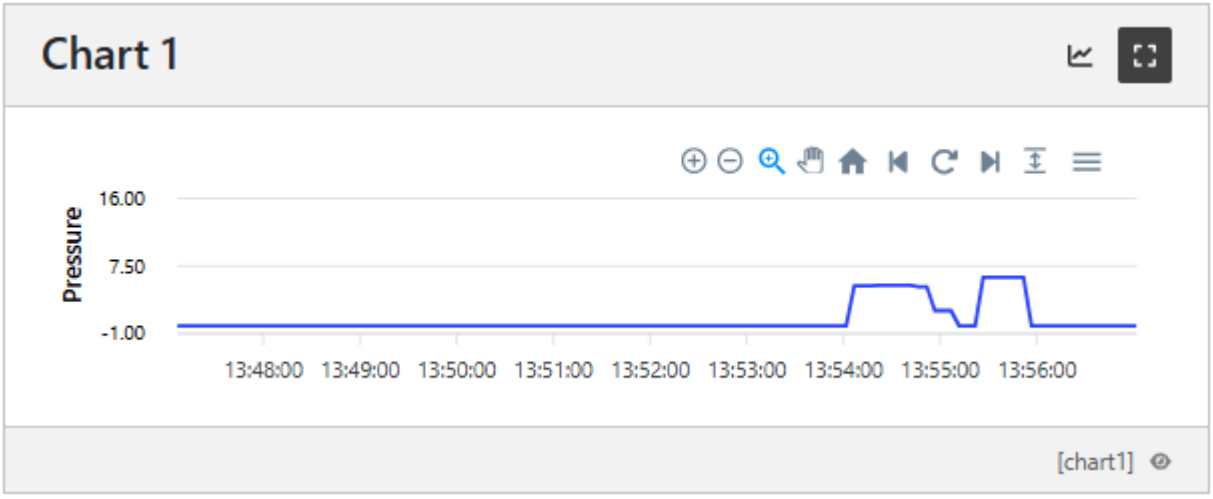


Figure 4 - Typical Pressure Chart Shown on Senquip Portal

The approach demonstrated for reading sensor data from BLE advertisement packets is applicable to any BLE sensor that encodes its measurements in the advertisement payload. The SQ.parse() function provides a concise way to extract and convert multi-byte values from both raw byte strings and hex strings, handling endianness and data type conversion in a single call.

8. Disclaimer

The information provided in this application note is intended for informational purposes only. Users should exercise caution and adhere to all relevant safety guidelines and regulations when deploying pressure monitoring equipment. By utilising the information provided, users acknowledge their understanding and acceptance of the associated risks. The authors and contributors disclaim any warranties, expressed or implied, regarding the accuracy or completeness of the information presented.

9. Appendix A – A Simple Python UDP Server

```
import logging
import socket

log = logging.getLogger('udp_server')


def udp_server(host='192.168.1.146', port=23):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    log.info("Listening on udp %s:%s" % (host, port))
    s.bind((host, port))
    while True:
        (data, addr) = s.recvfrom(128*1024)
        yield data


FORMAT_CONS = '%(asctime)s %(name)-12s %(levelname)8s\t%(message)s'
logging.basicConfig(level=logging.DEBUG, format=FORMAT_CONS)

for data in udp_server():
    log.debug("%r" % (data,))
```

10. Appendix B – Multiple Pressure Sensor Application

```
load("senquip.js");
load("api_events.js");
load("api_bt_gap.js");
load("api_timer.js");
load("api_endpoint.js");

// ---------------------------------------------------------------
// WLB-PT Pressure Sensor - multi-sensor advertisement decoder
// Decodes pressure from BLE advertisement data, no connection needed
// Pressure: bytes 10-13, little endian, signed 32-bit, microbar
// Divide by 10000 for kPa
// ---------------------------------------------------------------

// Add one entry per sensor - set addr to MAC address (uppercase, no colons)
// and cp to the desired custom parameter number
let sensor_map = {
  "D26DBB0A61BA": { cp: 1 },
  "AABBCCDDEEFF": { cp: 2 },  // replace with actual MAC address
};

let SCAN_TIME_MS = 2000;

function log(s) {
  UDP.send(s);
}

Event.on(GAP.EV_SCAN_RESULT, function(ev, evdata) {
  let sr = GAP.getScanResultArg(evdata);
  let s = sensor_map[sr.addr];

  // Ignore sensors not in the map
  if (!isdef(s)) {return;}

  // Bytes 10-13: pressure in microbar, little endian signed 32-bit
  // base=0 (raw bytes), -SQ.S32 reverses byte order for little endian
  let raw = SQ.parse(sr.advData, 10, 4, 0, -SQ.S32);
  let kpa = raw / 10000;

  SQ.dispatch(s.cp, kpa);

  log("addr=" + sr.addr +
      " cp=" + JSON.stringify(s.cp) +
      " kPa=" + JSON.stringify(kpa));
}, null);

Event.on(GAP.EV_SCAN_STOP, function() {
  GAP.scan(SCAN_TIME_MS, false, "WLB");
}, null);

log("Starting WLB-PT multi-sensor scan...");
GAP.scan(SCAN_TIME_MS, false, "WLB");
```