Decoder LTE – badge tracker & asset management tracker

Decoders · https://www.lansitec.com/es/docs/decoders/decoder-lte-badge-tracker/

decode-lte-badge&asset-tracker.jsDownload

Note: this decoder is made for MQTT protocol. If you want to integrate our devices with Thingsboard, http or other custom protocols, you can adapt the code below or contact us. We will support you and send you the specific decoder.

Below the full decoder code:

```
// Decode uplink function.
//
// Input is an object with the following fields:
// - bytes = Byte array containing the uplink payload, e.g. [255, 230, 255, 0]
// - fPort = Uplink fPort.
// - variables = Object containing the configured device variables.
//
// Output must be an object with the following fields:
// - data = Object representing the decoded payload.

// NB-IoT Badge Tracker
function decodeUplink(input) {
  // bytes
  var bytes = input.bytes;
  // type
  var uplinkType = (bytes[0] >> 4) & 0x0f;

  switch (uplinkType) {
    case 0x01:
      return { data: decodeRegistration(bytes) };

    case 0x02:
      return { data: decodeHeartbeat(bytes) };

    case 0x03:
      return { data: decodeGNSSPosition(bytes) };
```

```
    case 0x04:
      return { data: decodeBeacon(bytes) };

    case 0x05:
      return { data: decodeAlarm(bytes) };

    case 0x06:
      return { data: decodeConfigParameterResponse(bytes) };

    default:
      return;
  }
}

// type: 0x1 Registration
function decodeRegistration(bytes) {
  var data = {};
  data.type = "RegistrationMessage";
  data.bleEnable = (bytes[1] >> 7) & 0x01;
  data.gnssEnable = (bytes[1] >> 6) & 0x01;
  data.networkStatusCheckEnable = (bytes[1] >> 5) & 0x01;
  data.powerSwitchEnable = (bytes[1] >> 4) & 0x01;
  data.assetBeaconSortEnable = (bytes[1] >> 1) & 0x01;
  data.gnssFailureReportEnable = bytes[1] & 0x01;
  data.assetManagementEnable = (bytes[2] >> 7) & 0x01;

  var positionReportMode = (bytes[3] >> 6) & 0x0f;
  if (positionReportMode == 0) {
    data.positionReportMode = "Period";
  } else if (positionReportMode == 1) {
    data.positionReportMode = "Autonomous";
  } else if (positionReportMode == 2) {
    data.positionReportMode = "On-demand";
  }
  data.tamperDetectionEnable = (bytes[3] >> 3) & 0x01;

  data.heartbeatPeriod =
    (((bytes[6] << 8) & 0xff00) | (bytes[7] & 0xff)) * 30 + "s";

  data.blePositionReportInterval =
    (((bytes[8] << 8) & 0xff00) | (bytes[9] & 0xff)) * 5 + "s";

  data.blePositionBeaconReceivingDuration = bytes[10] & 0xff;
```

```
  data.gnssPositionReportInterval =
    (((bytes[11] << 8) & 0xff00) | (bytes[12] & 0xff)) * 5 + "s";

  data.gnssReceivingDuration = (bytes[13] & 0xff) * 5 + "s";

  data.assetBeaconReportInterval =
    (((bytes[14] << 8) & 0xff00) | (bytes[15] & 0xff)) * 5 + "s";

  data.assetBeaconReceivingDuration = bytes[16] & 0xff;

  data.version = ((bytes[17] << 8) & 0xff00) | (bytes[18] & 0xff);

  var imsi = "";
  for (let i = 0; i < 8; i++) {
    imsi += bytes[19 + i].toString(16).toUpperCase().padStart(2, "0");
  }
  data.imsi = imsi.substring(0, 15);
  data.messageId = ((bytes[27] << 8) & 0xff00) | (bytes[28] & 0xff);
  return data;
}

// type: 0x2 Heartbeat
function decodeHeartbeat(bytes) {
  var data = {};
  data.type = "HeartbeatMessage";
  var stateBitField = {};
  stateBitField.bleEnable = (bytes[1] >> 7) & 0x01;
  stateBitField.gnssEnable = (bytes[1] >> 6) & 0x01;
  stateBitField.networkStatusCheckEnable = (bytes[1] >> 5) & 0x01;
  stateBitField.powerSwitchEnable = (bytes[1] >> 4) & 0x01;
  stateBitField.assetBeaconSortEnable = (bytes[1] >> 1) & 0x01;
  stateBitField.gnssFailureReportEnable = bytes[1] & 0x01;
  stateBitField.assetManagementEnable = (bytes[2] >> 7) & 0x01;
  var positionReportMode = (bytes[3] >> 6) & 0x0f;
  if (positionReportMode == 0) {
    stateBitField.positionReportMode = "Period";
  } else if (positionReportMode == 1) {
    stateBitField.positionReportMode = "Autonomous";
  } else if (positionReportMode == 2) {
    stateBitField.positionReportMode = "On-demand";
  }
  stateBitField.tamperDetectionEnable = (bytes[3] >> 3) & 0x01;
  data.stateBitField = stateBitField;
```

```
  data.batteryVoltage = ((bytes[5] & 0xff) * 0.01 + 2.00) + "V";
  data.batteryLevel = (bytes[6] & 0xff) + "%";
  data.bleReceivingCount = bytes[7] & 0xff;
  data.gnssOnCount = bytes[8] & 0xff;
  // temperature
  if (0 == ((bytes[9] >> 7) & 0x01)) {
    data.temperature = (((bytes[9] << 8) & 0xff00) | (bytes[10] & 0xff)) + "°C";
  } else {
    data.temperature =
      (((bytes[9] << 8) & 0xff00) | (bytes[10] & 0xff)) * -1 + "°C";
  }
  data.movementDuration =
    (((bytes[11] << 8) & 0xff00) | (bytes[12] & 0xff)) * 5 + "s";
  data.chargeDuration = ((bytes[15] << 8) & 0xff00) | (bytes[16] & 0xff);
  data.messageId = ((bytes[17] << 8) & 0xff00) | (bytes[18] & 0xff);
  return data;
}

// type: 0x03 GNSSPosition
function decodeGNSSPosition(bytes) {
  var data = {};
  data.type = "GNSSPosition";
  // longitude
  let longitude =
    (bytes[1] << 24) | (bytes[2] << 16) | (bytes[3] << 8) | bytes[4];
  data.longitude = hex2float(longitude);

  // latitude
  let latitude =
    (bytes[5] << 24) | (bytes[6] << 16) | (bytes[7] << 8) | bytes[8];
  data.latitude = hex2float(latitude);

  // time
  let time =
    (bytes[9] << 24) | (bytes[10] << 16) | (bytes[11] << 8) | bytes[12];
  data.time = timestampToTime((time + 8 * 60 * 60) * 1000);

  return data;
}

// type: 0x4 Beacon
function decodeBeacon(bytes) {
  var data = {};
  data.type = "BeaconMessage";
```

```
  data.beaconType =
    (bytes[0] & 0x01) == 0 ? "PositioningBeacon" : "AssetBeacon";
  data.beaconCount = bytes[1] & 0x0f;
  for (let i = 0; i < data.beaconCount; i++) {
    var index = 2 + 5 * i;
    var major = (((bytes[index] << 8) & 0xff00) | (bytes[index + 1] & 0xff))
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var minor = (((bytes[index + 2] << 8) & 0xff00) | (bytes[index + 3] & 0xff))
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var rssi = bytes[index + 4] - 256 + "dBm";

    data["beacon" + (i + 1)] = major + minor;
    data["rssi" + (i + 1)] = rssi;
  }

  return data;
}

// type: 0x5 Alarm
function decodeAlarm(bytes) {
  var data = {};
  data.type = "AlarmMessage";
  var alarmValue = bytes[1] & 0x0f;
  if (alarmValue == 1) {
    data.alarm = "SOS";
  } else if (alarmValue == 2) {
    data.alarm = "Fall";
  }
  return data;
}

// type: 0x6 Configuration Parameter Response
function decodeConfigParameterResponse(bytes) {
  var data = {};
  data.type = "ConfigurationParameterResponse";
  var parameter = [];
  let byteLength = bytes.length;
  var index = 0;
  while (index + 1 < byteLength) {
    var commandBitField = {};
```

```
    var parameterType = bytes[index + 1] & 0xff;
    commandBitField.parameterType = parameterType;
    var commandBitFieldLength = getCommandBitFieldLength(parameterType);
    var parameterValue = getParameterValue(bytes, index, commandBitFieldLength);
    commandBitField.parameterValue = parameterValue;
    commandBitField.name = getParameterName(parameterType);
    commandBitField.parameterDefinition = getParameterDefinition(
      parameterType,
      parameterValue
    );
    index = index + commandBitFieldLength;
    parameter.push(commandBitField);
  }
  data.parameter = parameter;
  return data;
}

// getParameterValue hexString
function getParameterValue(bytes, index, length) {
  var hexString = "";
  for (var i = 2; i <= length; i++) {
    var hex = (bytes[index + i] & 0xff).toString(16).toUpperCase();
    hexString += hex.padStart(2, "0");
  }
  return hexString;
}

// getCommandBitFieldLength
function getCommandBitFieldLength(parameterType) {
  let lengths = {
    0x00: 3,
    0x01: 3,
    0x02: 3,
    0x03: 3,
    0x04: 3,
    0x05: 2,
    0x06: 2,
    0x07: 2,
    0x08: 2,
    0x0a: 17,
    0x0b: 17,
    0x0e: 9,
    0x1e: 2,
    0x1f: 2,
```

```
    0x20: 2,
    0x29: 2,
    0x2a: 2,
    0x2b: 2,
    0x2e: 2,
    0x2f: 2,
    0x30: 2,
    0x31: 2,
  };
  return lengths[parameterType] ?? 0;
}

// Parameter Name
function getParameterName(parameterType) {
  let name = {
    0x00: "SoftwareVersion",
    0x01: "HBPeriod",
    0x02: "BlePositionReportInterval",
    0x03: "GNSSPositionReportInterval",
    0x04: "AssetBeaconReportInterval",
    0x05: "BlePositionReceivingDuration",
    0x06: "GNSSPositionReceivingDuration",
    0x07: "AssetBleReceivingDuration",
    0x08: "FallThreshold",
    0x0a: "PosBeaconUUID",
    0x0b: "AssetBeaconUUID",
    0x0e: "ISMI",
    0x1e: "FallDetectionAlarmEnable",
    0x1f: "SOSAlarmEnable",
    0x20: "PosMode",
    0x29: "AssetManagementEnable",
    0x2a: "GNSSFailureReportEnable",
    0x2b: "AssetBeaconSortEnable",
    0x2e: "PowerSwitchEnable",
    0x2f: "NetworkStatusCheck",
    0x30: "GNSSEnableState",
    0x31: "BleEnable",
  };
  return name[parameterType] ?? "No matching parameter names";
}

// Parameter Definition
function getParameterDefinition(parameterType, parameterValue) {
  var val = parseInt(parameterValue, 16);
```

```
  let name = {
    0x00: "SoftwareVersion",
    0x01: val * 30 + "s, The interval of the heartbeat message, unit: 30s",
    0x02: val * 5 + "s, The interval of Bluetooth position repor, unit: 5s",
    0x03: val * 5 + "s, The interval of GNSS position report, unit: 5s",
    0x04: val * 5 + "s, The interval of asset beacons report, unit: 5s",
    0x05:
      val * 1 + "s, The duration of BLE position beacon receiving, unit: 1s",
    0x06: val * 5 + "s, The duration of GNSS position receiving, unit: 5s",
    0x07: val * 1 + "s, The duration of asset beacon receiving, unit 1s",
    0x08:
      val / 2 +
      "s, The threshold of the fall detection, the unit is 0.5 meters",
    0x0a: "PosBeaconUUIDFilter",
    0x0b: "AssetBeaconUUIDFilter",
    0x0e: "ISMI",
    0x1e: val == 0 ? "Disable" : "Enable",
    0x1f: val == 0 ? "Disable" : "Enable",
    0x20:
      val == 0
        ? "Period Mode"
        : val == 1
        ? " Autonomous Mode"
        : "On-demand Mode",
    0x29: val == 0 ? "Disable" : "Enable",
    0x2a: val == 0 ? "Disable" : "Enable",
    0x2b: val == 0 ? "Disable" : "Enable",
    0x2e: val == 0 ? "Disable" : "Enable",
    0x2f: val == 0 ? "Disable" : "Enable",
    0x30: val == 0 ? "Disable" : "Enable",
    0x31: val == 0 ? "Disable" : "Enable",
  };
  return name[parameterType] ?? "No matching parameter names";
}

// Floating point conversion
function hex2float(num) {
  var sign = num & 0x80000000 ? -1 : 1;
  var exponent = ((num >> 23) & 0xff) - 127;
  var mantissa = 1 + (num & 0x7fffff) / 0x7fffff;
  return sign * mantissa * Math.pow(2, exponent);
}

function asciiToHex(str) {
```

```
  var hexString = "";
  for (let i = 0; i < str.length; i++) {
    var hex = str.charCodeAt(i).toString(16);
    hexString += hex.padStart(2, "0");
  }
  return hexString;
}

function hexStringToByteArray(hexString) {
  var byteArray = [];
  for (let i = 0; i < hexString.length; i += 2) {
    byteArray.push(parseInt(hexString.substr(i, 2), 16));
  }
  return new Uint8Array(byteArray);
}

function timestampToTime(timestamp) {
  const date = new Date(timestamp);
  const year = date.getFullYear();
  const month = (date.getMonth() + 1).toString().padStart(2, "0");
  const day = date.getDate().toString().padStart(2, "0");
  const hour = date.getHours().toString().padStart(2, "0");
  const minute = date.getMinutes().toString().padStart(2, "0");
  const second = date.getSeconds().toString().padStart(2, "0");
  return `${year}-${month}-${day} ${hour}:${minute}:${second}`;
}
```