Decoder LTE – Bluetooth gateway

Decoders · https://www.lansitec.com/de/docs/decoders/lte-bluetooth-gateway-decoder/

decode-lte-bluetooth-gateway.jsDownload

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 configconsoleured device variables.
//
// Output must be an object with the following fields:
// - data = Object representing the decoded payload.

// NB-IoT Bluetooth Gateway
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 0x06:
      return { data: decodeConfigParameterResponse(bytes) };

    case 0x07:
      return { data: decodeBleDeviceMessage(bytes) };

    default:
      return;
  }
}
```

```
// type: 0x1 Registration
function decodeRegistration(bytes) {
  var data = {};
  data.type = "Registration";

  data.deviceRssiSortEnable = (bytes[1] >> 1) & 0x01;

  data.bleReceivingEnable = (bytes[2] >> 7) & 0x01;

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

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

  data.bleReceivingDuration = 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.deviceRssiSortEnable = (bytes[1] >> 1) & 0x01;
  stateBitField.bleReceivingEnable = (bytes[2] >> 7) & 0x01;
  data.stateBitField = stateBitField;

  data.batteryVoltage = (bytes[5] & 0xff) * 0.1 + "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: 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);
    if (commandBitFieldLength == 0x10) {
      var deviceReportRule = decodeDeviceReportRule(bytes, index);
      commandBitFieldLength = deviceReportRule.index;
      commandBitField.parameterValue = deviceReportRule;
      commandBitField.name = getParameterName(parameterType);
      commandBitField.parameterDefinition = "DeviceReportRule";
    } else {
      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;
}

// type: 0x7 BleDeviceMessage
function decodeBleDeviceMessage(bytes) {
  var data = {};
  data.type = "BleDeviceMessage";
  data.ruleType = bytes[0] & 0x0f;
  data.number = bytes[1] & 0x0f;
  for (let i = 0; i < data.number; 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;
}

// decodeDeviceReportRule
function decodeDeviceReportRule(bytes, reportIndex) {
  var data = {};
  var index = reportIndex + 2;
  data.type = "DeviceReportRule";
  data.deviceTypeQuantity = bytes[index++] & 0xff;
  data.deviceTypeId = (bytes[index] >> 4) & 0x0f;
  data.filterAndDataBlockQuantity = bytes[index++] & 0x0f;
  var filterBlock = [];
  var dataBlock = [];
  var macBlock = [];
  for (let i = 0; i < data.filterAndDataBlockQuantity; i++) {
    var ruleType = bytes[index++] & 0xff;
```

```
    var startAddress = bytes[index++] & 0xff;
    var endAddress = bytes[index++] & 0xff;
    var filter = {};
    if (ruleType == 1) {
      filter.ruleType = "FilterBlock";
      filter.startAddress = byteToHex(startAddress);
      filter.endAddress = byteToHex(endAddress);
      var len = endAddress - startAddress;
      var filterValue = "";
      for (let j = 0; j < len + 1; j++) {
        filterValue += byteToHex(bytes[index + j]);
      }
      filter.value = filterValue;
      index = index + (endAddress - startAddress + 1);
      filterBlock.push(filter);
    } else if (ruleType == 2) {
      filter.ruleType = "DataBlock";
      filter.startAddress = byteToHex(startAddress);
      filter.endAddress = byteToHex(endAddress);
      dataBlock.push(filter);
    } else if (ruleType == 3) {
      filter.ruleType = "MACBlock";
      filter.startAddress = byteToHex(startAddress);
      filter.endAddress = byteToHex(endAddress);
      macBlock.push(filter);
    }
  }
  data.filterBlock = filterBlock;
  data.dataBlock = dataBlock;
  data.macBlock = macBlock;
  data.index = index;
  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,
    0x04: 3,
    0x07: 2,
    0x10: 0x10,
    0x20: 2,
    0x29: 2,
    0x2b: 2,
  };
  return lengths[parameterType] ?? 0;
}

// Parameter Name
function getParameterName(parameterType) {
  let name = {
    0x00: "SoftwareVersion",
    0x01: "HBPeriod",
    0x04: "DeviceReportInterval",
    0x07: "BleReceivingDuration",
    0x10: "DeviceReportRule",
    0x29: "BleReceivingEnable",
    0x2b: "DeviceRssiSortEnable",
  };
  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",
    0x04: val * 5 + "s, The interval of BLE devices message report, unit: 5s",
    0x07: val * 1 + "s, The duration of Bluetooth receiving, unit 1s",
    0x29: val == 0 ? "Disable" : "Enable",
    0x2b: 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 byteToHex(str) {
  return str.toString(16).toUpperCase().padStart(2, "0");
}
```