Decoder LoRaWAN – bluetooth proximity gateway (TTN/Chirpstack)

Decoders · https://www.lansitec.com/pt/docs/decoders/decoder-lorawan-bluetooth-proximity-gateway-ttn-chirpstack/

decode-bluetooth-proximity-gateway-ttn_chirpstackv4.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 configured device variables.
//
// Output must be an object with the following fields:
// - data = Object representing the decoded payload.

// 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 0x03:
      return { data: decodeDeviceReportRule(bytes) };

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

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

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

    case 0x08:
      return { data: decodeDeviceType1(bytes) };

    case 0x09:
      return { data: decodeDeviceType2(bytes) };

    case 0x0a:
      return { data: decodeDeviceType3(bytes) };

    case 0x0b:
      return { data: decodeOptionalConfigurationReport(bytes) };

    case 0x0e:
      return { data: decodeMultiDeviceTypeMessage(bytes) };

    case 0x0f:
      return { data: decodeAcknowledgment(bytes) };

    default:
      return null;
  }
}

// type: 0x1 Registration
function decodeRegistration(bytes) {
  var data = {};
  data.type = "Registration";
  // adr
  data.adr = ((bytes[0] >> 3) & 0x1) == 0 ? "OFF" : "ON";
  // mode
  data.mode = bytes[0] & 0x07;
  // loRaWANBand
  var loRaWANBandValue = bytes[1];
  if (loRaWANBandValue == 0x00) {
    data.loRaWANBand = "KR920";
  } else if (loRaWANBandValue == 0x01) {
    data.loRaWANBand = "AU915";
  } else if (loRaWANBandValue == 0x04) {
    data.loRaWANBand = "CN470";
  } else if (loRaWANBandValue == 0x08) {
```

```
    data.loRaWANBand = "AS923";
  } else if (loRaWANBandValue == 0x10) {
    data.loRaWANBand = "EU433";
  } else if (loRaWANBandValue == 0x20) {
    data.loRaWANBand = "EU868";
  } else if (loRaWANBandValue == 0x40) {
    data.loRaWANBand = "US915";
  }
  // power
  data.power = ((bytes[2] >> 3) & 0x1f) + "dBm";
  // continuousBleReceiveEnable
  data.continuousBleReceiveEnable =
    ((bytes[2] >> 1) & 0x1) == 0 ? "Disable" : "Enable";
  // dr
  data.dr = (bytes[3] >> 4) & 0x0f;
  // DeviceType
  var DeviceTypeValue = (bytes[3] >> 1)& 0x0;
  if (DeviceTypeValue == 0x00) {
    data.DeviceType = "Solar gateway";
  } else if (DeviceTypeValue == 0x01) {
    data.DeviceType = "Compact gateway";
  } else if (DeviceTypeValue == 0x02) {
    data.DeviceType = "SocketSync gateway";
  } else if (DeviceTypeValue == 0x03) {
    data.DeviceType = "PA solar gateway";
  } else if (DeviceTypeValue == 0x04) {
    data.DeviceType = "Macro gateway";
  } else if (DeviceTypeValue == 0x05) {
    data.DeviceType = "Macro proximity gateway";
  } else if (DeviceTypeValue == 0x06) {
    data.DeviceType = "Micro gateway";
  }
  // positionReportInterval
  data.positionReportInterval =
    (((bytes[4] << 8) & 0xff00) | (bytes[5] & 0xff)) * 5 + "s";
  // heartbeatInterval
  data.heartbeatInterval = bytes[6] * 30 + "s";
  // bleReceivingDuration
  data.bleReceivingDuration = bytes[7] + "s";
  // networkReconnectionInterval
  data.networkReconnectionInterval = bytes[8] * 5 + "min";
  return data;
}
```

```
// type: 0x2 Heartbeat
function decodeHeartbeat(bytes) {
  var data = {};
  // type
  data.type = "Heartbeat";
  // battery
  var batteryValue = bytes[1];
  if (batteryValue > 100) {
    data.battery = bytes[1] / 100 + 1.5 + "V";
  } else {
    data.battery = bytes[1] + "%";
  }
  // rssi
  data.rssi = bytes[2] * -1 + "dBm";
  // snr
  data.snr = ((((bytes[3] << 8) & 0xff00) | (bytes[4] & 0xff))- 65536) * 0.01 + "dB";
  // version
  data.version = ((bytes[5] << 8) & 0xff00) | (bytes[6] & 0xff);
  // chargeState
  var chargeStateValue = bytes[7] & 0xff;
  if (chargeStateValue == 0x00) {
    data.chargeState = "Not charging";
  } else if (chargeStateValue == 0x50) {
    data.chargeState = "Charging";
  } else if (chargeStateValue == 0x60) {
    data.chargeState = "Charging completed";
  }
  return data;
}

// type: 0x3 DeviceReportRule
function decodeDeviceReportRule(bytes) {
  var data = {};
  data.type = "DeviceReportRule";
  data.deviceTypeQuantity = bytes[1] & 0xff;
  data.deviceTypeId = (bytes[2] >> 4) & 0x0f;
  data.filterAndDataBlockQuantity = bytes[2] & 0x0f;
  var filterBlock = [];
  var dataBlock = [];
  var macBlock = [];
  var macfilterBlock = [];
  var index = 3;
  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);
  } else if (ruleType == 4) {
    filter.ruleType = "MACFilterBlock";
    filter.startAddress = byteToHex(startAddress);
    filter.endAddress = byteToHex(endAddress);
    var len = endAddress - startAddress;
    var macfilterValue = "";
    for (let j = 0; j < len + 1; j++) {
      macfilterValue += byteToHex(bytes[index + j]);
    }
    filter.value = macfilterValue;
    index = index + (endAddress - startAddress + 1);
    macfilterBlock.push(filter);
  }
}
data.filterBlock = filterBlock;
data.dataBlock = dataBlock;
data.macBlock = macBlock;
data.macfilterBlock = macfilterBlock;
return data;
```

```
}

// type: 0x04 GNSSPosition
function decodeGNSSPosition(bytes) {
  var data = {};
  data.type = "GNSSPosition";
  // gnssState
  data.gnssState = (bytes[0] & 0x01) == 0 ? "Success" : "Fail";
  // 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: 0x5 WaterLevelDetection
function decodeWaterLevelDetection(bytes) {
var data = {};
// type
data.type = "WaterLevelDetection";
data.waterLevel = (((bytes[1] << 8) & 0xff00) | (bytes[2] & 0xff)) + "mm";
return data;
}

// type: 0x7 PositionBeacon
function decodePositionBeacon(bytes) {
  var data = {
    type: "PositionBeacon",
    number: bytes[1] & 0x0f,
  };
  var index = 1;
  for (let i = 0; i < data.number; i++) {
    var major = ((bytes[index + 1] << 8) | bytes[index + 2])
      .toString(16)
      .toUpperCase()
```

```
      .padStart(4, "0");
    var minor = ((bytes[index + 3] << 8) | bytes[index + 4])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var rssi = bytes[index + 5] - 256 + "dBm";

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

    index = index + 5;
  }

  return data;
}

// type: 0x8 DeviceType1
function decodeDeviceType1(bytes) {
  var data = {
    type: "DeviceType1",
    number: bytes[0] & 0x0f,
  };
  var index = 1;
  for (let i = 0; i < data.number; i++) {
    var major = ((bytes[index] << 8) | bytes[index + 1])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var minor = ((bytes[index + 2] << 8) | bytes[index + 3])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var rssi = bytes[index + 4] - 256 + "dBm";

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

    index = index + 5;
  }

  return data;
}

// type: 0x9 DeviceType2
```

```
function decodeDeviceType2(bytes) {
  var data = {
    type: "DeviceType2",
    number: bytes[0] & 0x0f,
  };
  var index = 1;
  for (let i = 0; i < data.number; i++) {
    var major = ((bytes[index] << 8) | bytes[index + 1])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var minor = ((bytes[index + 2] << 8) | bytes[index + 3])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var rssi = bytes[index + 4] - 256 + "dBm";

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

    index = index + 5;
  }

  return data;
}

// type: 0xa DeviceType3
function decodeDeviceType3(bytes) {
  var data = {
    type: "DeviceType3",
    number: bytes[0] & 0x0f,
  };
  var index = 1;
  for (let i = 0; i < data.number; i++) {
    var major = ((bytes[index] << 8) | bytes[index + 1])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var minor = ((bytes[index + 2] << 8) | bytes[index + 3])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var rssi = bytes[index + 4] - 256 + "dBm";
```

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

    index = index + 5;
  }

  return data;
}

// type: 0xe MultiDeviceTypeMessage
function decodeMultiDeviceTypeMessage(bytes) {
  var data = {
    type: "MultiDeviceTypeMessage",
    number: bytes[0] & 0x0f,
  };
  var index = 1;
  for (let i = 0; i < data.number; i++) {
    var index = 1 + 6 * i;
    var deviceTypeId = bytes[index];
    var major = ((bytes[index + 1] << 8) | bytes[index + 2])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var minor = ((bytes[index + 3] << 8) | bytes[index + 4])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var rssi = bytes[index + 5] - 256 + "dBm";

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

    index = index + 6;
  }

  return data;
}

// type: 0xf Acknowledgment
function decodeAcknowledgment(bytes) {
  var data = {};
  data.type = "Acknowledgment";
  data.result = (bytes[0] & 0x0f) == 0 ? "Success" : "Failure";
```

```
  data.msgId = (bytes[1] & 0xff).toString(16).toUpperCase();

  return data;
}

// type: 0xb OptionalConfigurationReport
function decodeOptionalConfigurationReport(bytes){
  var data = {
    type: "OptionalConfigurationReport",
  };
  var optionalParametersType = bytes[1] & 0xff;
  if (optionalParametersType == 0x01) {
    data.parametersType = "GNSS position report interval";
    data.parametersValue = ((bytes[2] << 8) | bytes[3]) * 5 + " minutes";
  }
  else if (optionalParametersType == 0x02) {
    data.parametersType = "Water level report interval";
    data.parametersValue = ((bytes[2] << 8) | bytes[3]) * 5 + " minutes";
  }
  else if (optionalParametersType == 0x03) {
    data.parametersType = "Duty-cycle configuration";
    data.parametersValue = ((bytes[2] << 8) | bytes[3]) * 0.1 + "%";
  }
  else if (optionalParametersType == 0x04) {
    data.parametersType = "Beacon reporting count configuration";
    data.parametersValue = (bytes[2] << 8) | bytes[3];
  }
  else if (optionalParametersType == 0x0f) {
    data.parametersType = "Ble rssi threshold";
    data.parametersValue = (bytes[2] - 256) + "dBm";
  }
  else if (optionalParametersType == 0x10) {
    data.parametersType = "Switch on/off time";
    data.wakesuptime = byteToDecimalStr(bytes[2]) + ":" + byteToDecimalStr(bytes[3]);
    data.shutdowntime = byteToDecimalStr(bytes[4]) + ":" + byteToDecimalStr(bytes[5]);
  }

  return data;
}

function byteToHex(str) {
  return str.toString(16).toUpperCase().padStart(2, "0");
}
```

```
function byteToDecimalStr(byte) {
  return byte.toString().padStart(2, '0');
}

function hex2float(num) {
  const sign = (num >>> 31) ? -1 : 1;
  const exponent = ((num >>> 23) & 0xff);
  const mantissa = num & 0x7fffff;

  if (exponent === 0) {
    // Subnormal number
    return sign * Math.pow(2, -126) * (mantissa / Math.pow(2, 23));
  } else if (exponent === 255) {
    if (mantissa === 0) return sign * Infinity;
    else return NaN;
  } else {
    // Normalized number
    return sign * Math.pow(2, exponent - 127) * (1 + mantissa / Math.pow(2, 23));
  }
}

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}`;
}
```