Decoder LoRaWAN – UWB tracker (TTN/Chirpstack)

Decoders · https://www.lansitec.com/pt/docs/decoders/decoder-lorawan-uwb-tracker-ttn-chirpstack/

decode-uwb-tracker.jsDownload

Below the full decoder code:

```
// UWB
function decodeUplink(input) {
  // bytes
  var bytes = input.bytes;
  // type
  var type = (bytes[0] >> 4) & 0x0f;

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

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

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

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

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

    default:
      return;
  }
}

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

```
  data.softwareVersion = "V" + (((bytes[1] << 8) & 0xff00) | (bytes[2] &
0xff)).toString(16).toUpperCase().padStart(4, "0");
  data.hardwareVersion = "V" + (((bytes[3] << 8) & 0xff00) | (bytes[4] &
0xff)).toString(16).toUpperCase().padStart(4, "0");
  data.loraClockSyncInterval = (((bytes[9] << 8) & 0xff00) | (bytes[10] & 0xff)) + "s";
  data.loraClockSyncFreq = ((bytes[11] << 24) | (bytes[12] << 16) | (bytes[13] << 8) |
bytes[14]) / 1000000;
  data.loraClockSyncSF = bytes[15];
  var loraClockSyncBWValue = bytes[16];
  if (loraClockSyncBWValue == 0) {
    data.loraClockSyncBW = "500kHz";
  } else if (loraClockSyncBWValue == 1) {
    data.loraClockSyncBW = "250kHz";
  } else if (loraClockSyncBWValue == 2) {
    data.loraClockSyncBW = "125kHz";
  }
  data.heartbeatInterval = (((bytes[17] << 8) & 0xff00) | (bytes[18] & 0xff)) * 30 + "s";

  return data;
}

// type: 0x2 Heartbeat
function decodeHeartbeat(bytes) {
  var data = {};
  data.type = "HeartbeatMessage";
  data.batteryVoltage = (bytes[5] * 0.1).toFixed(1) + "V";
  data.batteryPercentage = bytes[6] + "%";
  data.uwbPositioningCount = bytes[7];
  data.temperature = (((bytes[10] << 8) & 0xff00) | (bytes[11] & 0xff)) + "°C";
  data.chargeTime = (((bytes[16] << 8) & 0xff00) | (bytes[17] & 0xff)) * 30 + "s";

  return data;
}

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

```
  return data;
}

// type: 0x6 Single Configuration Parameter
function decodeSingleConfigurationParameter(bytes) {
  var data = {};
  data.type = "ConfigurationParameterResponse";
  var parameter = [];
  let byteLength = bytes.length;
  var index = 0;
  while (index + 1 < byteLength) {
    var commandBitField = {};
    var parameterTypevalue = bytes[index + 1] & 0xff;
    commandBitField.parameterType = parameterTypevalue.toString(16).toUpperCase().padStart(2,
"0");
    var commandBitFieldLength = getCommandBitFieldLength(parameterTypevalue);
    var parameterValue = getParameterValue(bytes, index, commandBitFieldLength);
    commandBitField.parameterValue = parameterValue;
    commandBitField.name = getParameterName(parameterTypevalue);
    commandBitField.parameterDefinition = getParameterDefinition(
      parameterTypevalue,
      parameterValue
    );
    index = index + commandBitFieldLength;
    parameter.push(commandBitField);
  }
  data.parameter = parameter;
  return data;
}

// type: 0x8 Distance
function decodeDistance(bytes) {
  var data = {};
  data.type = "Distance";
  data.uwbAnchorQuantity = bytes[1] & 0x0f;
  for (let i = 0; i < data.uwbAnchorQuantity; i++) {
    var index = 2 + 7 * i;
    var dev = (((bytes[index] << 8) & 0xff00) | (bytes[index + 1] & 0xff))
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    var eui = (((bytes[index + 2] << 8) & 0xff00) | (bytes[index + 3] & 0xff))
      .toString(16)
      .toUpperCase()
```

```
      .padStart(4, "0");

    data["deveui" + (i + 1)] = dev + eui;
    data["distance" + (i + 1)] =
      ((bytes[index + 4] << 16) | (bytes[index + 5] << 8) | bytes[index + 6]) + "cm";
  }

  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,
    0x08: 2,
    0x32: 2,
    0x33: 2,
    0x52: 3,
    0x53: 5,
    0x54: 2,
    0x55: 2,
    0x61: 3,
  };
  return lengths[parameterType] ?? 0;
}

// Parameter Name
function getParameterName(parameterType) {
  let name = {
    0x00: "SoftwareVersion",
    0x01: "HBPeriod",
    0x08: "FallDetectionThreshold",
    0x32: "CFMMSG",
```

```
    0x33: "HBCOUNT",
    0x52: "LoRaCLKSYNCInterval",
    0x53: "LoRaCLKSYNCFrequency",
    0x54: "LoRaCLKSYNCSF",
    0x55: "LoRaCLKSYNCBW",
    0x61: "PositionReportInterval",
  };
  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",
    0x08: val * 0.5 + "m, The threshold of the fall detection, unit: 0.5m",
    0x32: val + " The interval of messages that must be acknowledged",
    0x33: val + " The number of heartbeat ACK that the tracker misses",
    0x52: val + "s, LoRa clock synchronization interval, unit: 1s",
    0x53: val + " LoRa clock synchronization frequency",
    0x54: val + " LoRa clock synchronization spreading factor",
    0x55: val + " LoRa clock synchronization bandwidth",
    0x61: val + "s, The tracker's UWB position report interval, unit: 1s",
  };
  return name[parameterType] ?? "No matching parameter names";
}

// Floating point conversion
function hex2float(num) {
  num = num >>> 0;

  var sign = (num & 0x80000000) ? -1 : 1;
  var exponent = (num >>> 23) & 0xff;
  var fraction = num & 0x7fffff;

  if (exponent === 0xff) {
    return fraction === 0 ? sign * Infinity : NaN;
  }

  if (exponent === 0) {
    return fraction === 0 ? sign * 0 : sign * Math.pow(2, -126) * (fraction / 0x800000);
  }
```

```
  return sign * Math.pow(2, exponent - 127) * (1 + fraction / 0x800000);
}

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}`;
}
```