Decoder LoRaWAN – asset management tracker (TTN/Chirpstack)

Decoders · https://www.lansitec.com/es/docs/decoders/decode-asset-management-tracker-ttn_chirpstackv4/

decode-asset-management-tracker-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.

// Asset Management 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 0x07:
      return { data: decodeBeacon(bytes) };

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

```
    default:
      return null;
  }
}

// type: 0x01 Registration
function decodeRegistration(bytes) {
  var data = {};
  data.type = "Registration";
  data.adr = ((bytes[0] >> 3) & 0x1) == 0 ? "OFF" : "ON";
  data.power = ((bytes[2] >> 3) & 0x1f) + "dBm";
  data.dr = (bytes[3] >> 4) & 0x0f;
  data.gnssEnable = ((bytes[3] >> 3) & 0x01) == 0 ? "Disable" : "Enable";
  var positionModeValue = (bytes[3] >> 1) & 0x03;
  if (positionModeValue == 0) {
    data.positionMode = "Period";
  } else if (positionModeValue == 1) {
    data.positionMode = "Autonomous";
  } else if (positionModeValue == 2) {
    data.positionMode = "Demand";
  }
  data.bleEnable = (bytes[3] & 0x01) == 0 ? "Disable" : "Enable";
  data.blePositionReportInterval =
    (((bytes[4] << 8) & 0xff00) | (bytes[5] & 0xff)) * 5 + "s";
  data.gnssPositionReportInterval =
    (((bytes[6] << 8) & 0xff00) | (bytes[7] & 0xff)) * 5 + "s";
  data.heartbeatReportInterval = (bytes[8] & 0xff) * 30 + "s";
  data.version =
    (bytes[9] & 0xff).toString(16).toUpperCase() +
    "." +
    (bytes[10] & 0xff).toString(16).toUpperCase();
  data.cfmsg = "1 Confirmed every " + (bytes[11] & 0xff) + " Heartbeat";
  data.hbCount = "Disconnect Judgement " + (bytes[12] & 0xff);
  data.fallDetection = (bytes[13] & 0xff) * 0.5 + " meters";
  return data;
}

// type: 0x02 Heartbeat
function decodeHeartbeat(bytes) {
  var data = {};
  data.type = "Heartbeat";
  data.battery = bytes[1] + "%";
  data.rssi = bytes[2] * -1 + "dBm";
```

```
  data.snr = (((bytes[3] << 8) & 0xff00) | (bytes[4] & 0xff)) / 100 + "dB";
  var gnssStateValue = (bytes[5] >> 4) & 0x0f;
  if (gnssStateValue == 0) {
    data.gnssState = "Off";
  } else if (gnssStateValue == 1) {
    data.gnssState = "Boot GNSS";
  } else if (gnssStateValue == 2) {
    data.gnssState = "Locating";
  } else if (gnssStateValue == 3) {
    data.gnssState = "Located";
  } else if (gnssStateValue == 9) {
    data.gnssState = "No signal";
  }
  data.moveState = bytes[5] & 0x0f;
  var chargeStateValue = (bytes[6] >> 4) & 0x0f;
  if (chargeStateValue == 0) {
    data.chargeState = "Power cable disconnected";
  } else if (chargeStateValue == 5) {
    data.chargeState = "Charging";
  } else if (chargeStateValue == 6) {
    data.chargeState = "Charge complete";
  }
  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: 0x07 Beacon
function decodeBeacon(bytes) {
  const data = {
    type: "Beacon",
    length: bytes[0] & 0x0f,
  };

  for (let i = 0; i < data.length; i++) {
    const index = 6 + 5 * i;
    const major = ((bytes[index] << 8) | bytes[index + 1])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    const minor = ((bytes[index + 2] << 8) | bytes[index + 3])
      .toString(16)
      .toUpperCase()
      .padStart(4, "0");
    const rssi = bytes[index + 4] - 256 + "dBm";

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

  return data;
}

// type: 0x08 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";
  } else {
    data.alarm = "Unknown";
  }

  return data;
```

```
}

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 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}`;
}
```