Decoder NTN – Livestock solar eartag tracker

Decoders · https://www.lansitec.com/es/docs/decoders/decoder-ntn-livestock-solar-eartag-tracker/

decode-livestock-tag-tracker-ntn-udpDownload

To create automatically uplink and downlink messages, please refer to this page:

Lansitec Decoder – Live Calculator – Lansitec

Below the full decoder code:

```
// Lansitec NTN Livestock Tracking Tag - NTN UDP Status Report decoder.
//
// The UDP server must pass the raw 21-byte datagram payload, not a printable
// hexadecimal string:
//   decodeNtn({ bytes: [...] })
//   decodeNtn(Buffer.from(...))

"use strict";

function decodeNtn(input) {
  var normalized = normalizeBytes(input);
  if (normalized.error) {
    return errorResult(normalized.error);
  }

  var bytes = normalized.bytes;
  if (bytes.length !== 21) {
    return errorResult("NTN Status Report must be exactly 21 bytes; received " + bytes.length +
".");
  }

  if (bytes[0] !== 0x01) {
    return errorResult("Unsupported NTN payload version: " + toHex(bytes[0], 2) + ".");
  }

  var batteryRaw = bytes[5];
  if (batteryRaw > 100 && batteryRaw !== 0xff) {
    return errorResult("NTN battery value must be between 0 and 100, or 0xFF when
```

```
unavailable.");
  }

  var alarmRaw = bytes[8];
  if ((alarmRaw & 0xf0) !== 0) {
    return errorResult("NTN Alarm Flags RFU bits 7 to 4 must be zero.");
  }

  var longitude = readFloat32BE(bytes, 9);
  var latitude = readFloat32BE(bytes, 13);
  var positionAvailable = !(longitude === 0 && latitude === 0);

  if (!isFiniteNumber(longitude) || longitude < -180 || longitude > 180) {
    return errorResult("NTN longitude must be a finite value in the range -180 to 180
degrees.");
  }
  if (!isFiniteNumber(latitude) || latitude < -90 || latitude > 90) {
    return errorResult("NTN latitude must be a finite value in the range -90 to 90 degrees.");
  }

  var timestamp = readUInt32BE(bytes, 17);
  var activeAlarms = [];
  if ((alarmRaw & 0x01) !== 0) activeAlarms.push("lowBattery");
  if ((alarmRaw & 0x02) !== 0) activeAlarms.push("fenceBreach");
  if ((alarmRaw & 0x04) !== 0) activeAlarms.push("abnormalInactivity");
  if ((alarmRaw & 0x08) !== 0) activeAlarms.push("highIntensityMotion");

  var data = {
    transport: "NTN_UDP",
    messageName: "statusReport",
    payloadVersion: bytes[0],
    deviceId: toHex(readUInt32BE(bytes, 1), 8),
    deviceIdNumeric: readUInt32BE(bytes, 1),
    batteryAvailable: batteryRaw !== 0xff,
    batteryPercent: batteryRaw === 0xff ? null : batteryRaw,
    motionDurationSeconds: readUInt16BE(bytes, 6),
    alarmFlags: {
      raw: alarmRaw,
      lowBattery: (alarmRaw & 0x01) !== 0,
      fenceBreach: (alarmRaw & 0x02) !== 0,
      abnormalInactivity: (alarmRaw & 0x04) !== 0,
      highIntensityMotion: (alarmRaw & 0x08) !== 0,
      active: activeAlarms,
    },
```

```
    positionAvailable: positionAvailable,
    longitude: positionAvailable ? longitude : null,
    latitude: positionAvailable ? latitude : null,
    timestampAvailable: timestamp !== 0,
    timestamp: timestamp === 0 ? null : timestamp,
    timestampIso: timestamp === 0 ? null : unixTimeToIso(timestamp),
    payloadHex: bytesToHex(bytes),
  };

  return { data: data };
}

function normalizeBytes(input) {
  var source = input && typeof input === "object" && input.bytes !== undefined ? input.bytes :
input;
  if (!source || typeof source === "string" || typeof source.length !== "number") {
    return { error: "Input must be a raw byte array, Uint8Array, or Buffer. Plain-text HEX is
not accepted." };
  }

  var bytes = [];
  for (var i = 0; i < source.length; i++) {
    var value = source[i];
    if (typeof value !== "number" || !isFinite(value) || Math.floor(value) !== value || value <
0 || value > 255) {
      return { error: "Payload byte at index " + i + " is not an unsigned 8-bit integer." };
    }
    bytes.push(value);
  }
  return { bytes: bytes };
}

function readUInt16BE(bytes, offset) {
  return bytes[offset] * 0x100 + bytes[offset + 1];
}

function readUInt32BE(bytes, offset) {
  return (
    bytes[offset] * 0x1000000 +
    bytes[offset + 1] * 0x10000 +
    bytes[offset + 2] * 0x100 +
    bytes[offset + 3]
  );
}
```

```
function readFloat32BE(bytes, offset) {
  var bits = readUInt32BE(bytes, offset);
  var sign = bits >= 0x80000000 ? -1 : 1;
  var exponent = Math.floor(bits / 0x800000) & 0xff;
  var fraction = bits & 0x7fffff;

  if (exponent === 0xff) {
    return fraction === 0 ? sign * Infinity : NaN;
  }
  if (exponent === 0) {
    return sign * fraction * Math.pow(2, -149);
  }
  return sign * (1 + fraction / 0x800000) * Math.pow(2, exponent - 127);
}

function unixTimeToIso(seconds) {
  return new Date(seconds * 1000).toISOString();
}

function bytesToHex(bytes) {
  var parts = [];
  for (var i = 0; i < bytes.length; i++) {
    parts.push(toHex(bytes[i], 2).slice(2));
  }
  return parts.join("");
}

function isFiniteNumber(value) {
  return typeof value === "number" && isFinite(value);
}

function toHex(value, width) {
  var text = Number(value).toString(16).toUpperCase();
  while (text.length < width) {
    text = "0" + text;
  }
  return "0x" + text;
}

function errorResult(message) {
  return { errors: [message] };
}
```

```
if (typeof module !== "undefined" && module.exports) {
  module.exports = {
    decodeNtn: decodeNtn,
  };
}
```