Payload Decoder DEOS SAM TTN V3

```
function cbor_decode(data, tagger, simpleValue) {
  var dataView = new DataView(data);
  var offset = 0;
  var POW_2_24 = 5.960464477539063e-8,
  POW_2_32 = 4294967296,
  POW_2_53 = 9007199254740992;

  if (typeof tagger !== "function")
    tagger = function (value) { return value; };
  if (typeof simpleValue !== "function")
    simpleValue = function () { return undefined; };

  function commitRead(length, value) {
    offset += length;
    return value;
  }
  function readArrayBuffer(length) {
    return commitRead(length, new Uint8Array(data, offset, length));
  }
  function readFloat16() {
    var tempArrayBuffer = new ArrayBuffer(4);
    var tempDataView = new DataView(tempArrayBuffer);
    var value = readUint16();

    var sign = value & 0x8000;
    var exponent = value & 0x7c00;
    var fraction = value & 0x03ff;

    if (exponent === 0x7c00)
      exponent = 0xff << 10;
    else if (exponent !== 0)
      exponent += (127 - 15) << 10;
    else if (fraction !== 0)
      return (sign ? -1 : 1) * fraction * POW_2_24;

    tempDataView.setUint32(0, sign << 16 | exponent << 13 | fraction << 13);
    return tempDataView.getFloat32(0);
  }
  function readFloat32() {
    return commitRead(4, dataView.getFloat32(offset));
  }
  function readFloat64() {
    return commitRead(8, dataView.getFloat64(offset));
  }
  function readUint8() {
```

```
    return commitRead(1, dataView.getUint8(offset));
}
function readUint16() {
    return commitRead(2, dataView.getUint16(offset));
}
function readUint32() {
    return commitRead(4, dataView.getUint32(offset));
}
function readUint64() {
    return readUint32() * POW_2_32 + readUint32();
}
function readBreak() {
    if (dataView.getUint8(offset) !== 0xff)
        return false;
    offset += 1;
    return true;
}
function readLength(additionalInformation) {
    if (additionalInformation < 24)
        return additionalInformation;
    if (additionalInformation === 24)
        return readUint8();
    if (additionalInformation === 25)
        return readUint16();
    if (additionalInformation === 26)
        return readUint32();
    if (additionalInformation === 27)
        return readUint64();
    if (additionalInformation === 31)
        return -1;
    throw "Invalid length encoding";
}
function readIndefiniteStringLength(majorType) {
    var initialByte = readUint8();
    if (initialByte === 0xff)
        return -1;
    var length = readLength(initialByte & 0x1f);
    if (length < 0 || (initialByte >> 5) !== majorType)
        throw "Invalid indefinite length element";
    return length;
}

function appendUtf16Data(utf16data, length) {
    for (var i = 0; i < length; ++i) {
        var value = readUint8();
        if (value & 0x80) {
            if (value < 0xe0) {
                value = (value & 0x1f) << 6
                    | (readUint8() & 0x3f);
                length -= 1;
            } else if (value < 0xf0) {
                value = (value & 0x0f) << 12
```

```
                | (readUint8() & 0x3f) << 6
                | (readUint8() & 0x3f);
            length -= 2;
          } else {
            value = (value & 0x0f) << 18
                | (readUint8() & 0x3f) << 12
                | (readUint8() & 0x3f) << 6
                | (readUint8() & 0x3f);
            length -= 3;
          }
        }

        if (value < 0x10000) {
          utf16data.push(value);
        } else {
          value -= 0x10000;
          utf16data.push(0xd800 | (value >> 10));
          utf16data.push(0xdc00 | (value & 0x3ff));
        }
      }
    }

    function decodeItem() {
      var initialByte = readUint8();
      var majorType = initialByte >> 5;
      var additionalInformation = initialByte & 0x1f;
      var i;
      var length;

      if (majorType === 7) {
        switch (additionalInformation) {
          case 25:
            return readFloat16();
          case 26:
            return readFloat32();
          case 27:
            return readFloat64();
        }
      }

      length = readLength(additionalInformation);
      if (length < 0 && (majorType < 2 || 6 < majorType))
        throw "Invalid length";

      switch (majorType) {
        case 0:
          return length;
        case 1:
          return -1 - length;
        case 2:
          if (length < 0) {
            var elements = [];
```

```
        var fullArrayLength = 0;
        while ((length = readIndefiniteStringLength(majorType)) >= 0) {
          fullArrayLength += length;
          elements.push(readArrayBuffer(length));
        }
        var fullArray = new Uint8Array(fullArrayLength);
        var fullArrayOffset = 0;
        for (i = 0; i < elements.length; ++i) {
          fullArray.set(elements[i], fullArrayOffset);
          fullArrayOffset += elements[i].length;
        }
        return fullArray;
      }
      return readArrayBuffer(length);
    case 3:
      var utf16data = [];
      if (length < 0) {
        while ((length = readIndefiniteStringLength(majorType)) >= 0)
          appendUtf16Data(utf16data, length);
      } else
        appendUtf16Data(utf16data, length);
      return String.fromCharCode.apply(null, utf16data);
    case 4:
      var retArray;
      if (length < 0) {
        retArray = [];
        while (!readBreak())
          retArray.push(decodeItem());
      } else {
        retArray = new Array(length);
        for (i = 0; i < length; ++i)
          retArray[i] = decodeItem();
      }
      return retArray;
    case 5:
      var retObject = {};
      for (i = 0; i < length || length < 0 && !readBreak(); ++i) {
        var key = decodeItem();
        retObject[key] = decodeItem();
      }
      return retObject;
    case 6:
      return tagger(decodeItem(), length);
    case 7:
      switch (length) {
        case 20:
          return false;
        case 21:
          return true;
        case 22:
          return null;
        case 23:
```

```
                return undefined;
              default:
                return simpleValue(length);
            }
        }
    }

    var ret = decodeItem();
    if (offset !== data.byteLength)
        throw "Remaining bytes";
    return ret;
}

function toArrayBuffer(buf) {
    var ab = new ArrayBuffer(buf.length);
    var view = new Uint8Array(ab);
    for (var i = 0; i < buf.length; ++i) {
        view[i] = buf[i];
    }
    return ab;
}

function sam_decode(data) {

    var float_temp = 0.0;
    var float_humi = 0.0;
    var dez_co2 = 0;

    data.forEach(function (element, index) {

        if (element[0] == 1) {
            float_temp = element[1];
        }
        else if (element[0] == 2) {
            float_humi = element[1];
        }
        else if (element[0] == 3) {
            dez_co2 = element[1];
        }
    });

    return { data: { "temperature": float_temp, "humidity": float_humi, "co2": dez_co2 } };
}

function decodeUplink(input) {

    var buffer = toArrayBuffer(input.bytes);
    var data = cbor_decode(buffer);
    var result = sam_decode(data);

    return result;
}
```

```
/*
Demo data

// TTN
// 8583011866018201FA41B800008202FA4274CCCD820318DD820400

var data = {
   bytes : [133, 131, 1, 24, 102, 1, 130, 1, 250, 65, 184, 0, 0, 130, 2, 250, 66, 116, 204, 205, 130, 3, 24, 221, 130, 4, 0]
};
var result = decodeUplink(data);
console.log(result);
*/
```