```
// Decoder for Mutelcor LoRaButton payload

var keywords = false; // For machine parsing set keywords to true, this will give keywords instead of text descriptions

var fahrenheit = false; // Set to true for temperature in Farenheit, otherwise Celcius

var uid_type_prefix = true; // Add the UID type byte value before the UID to make the type part of the UID


// TTN v3

function decodeUplink(input) {
  // Decode an uplink message from a buffer (array) of bytes to an object of fields.
  var decoded = MutelcorLoRaButtonDecode(input.bytes, input.fPort);


  var errors = [];
  if (decoded.error !== undefined) {
    errors.push(decoded.error);


    // Also add error description
    errors.push(translation.error_val[decoded.error]);
  }
  if (!keywords) decoded = Descriptions(decoded);
  return {
    data: decoded,
    warnings: [],
    errors: errors,
  };
}


// Chirpstack

function Decode(fPort, bytes, variables) {
  // Decode an uplink message from a buffer (array) of bytes to an object of fields.
  var decoded = MutelcorLoRaButtonDecode(bytes, fPort);
  if (!keywords) decoded = Descriptions(decoded);
```

```
  return decoded;
}

var translation = {
  port: "[01] Port",
  version: "[02] Payload Version",
  voltage: "[03] Voltage Battery/Input [V]",
  opcode: "[04] OpCode",
  opcode_val: {
    0: "Heartbeat",
    1: "Alarm",
    2: "Votes",
    3: "Measurements",
    4: "Location",
    5: "Thresholds",
    6: "Switch",
    7: "Reminder",
    80: "Feedback",
    112: "Info",
    113: "Show",
    114: "Update",
    128: "SCD30",
  },
  opcode_noval: "Unknown OpCode (@)",
  buttons: "[05] Buttons",
  totals: "[06] Button Totals",
  counts: "[07] Button Counts",
  qcrc: "[08] Questions CRC",
  meas: "[09] Measurements",
  temp: "[01] Temperature [°C]",
  temp_f: "[01] Temperature [°F]",
  rh: "[02] Relative Humidity [%]",
```

press: "[03] Pressure [hPa]",

light: "[04] Light [lx]",

co2: "[05] CO_2 [ppm]",

tvoc: "[06] Total Volatile Organic Compound [ppb]",

dist: "[07] Distance [mm]",

dinputs: "[08] Digital Inputs",

trigger: "[10] Thresholds Triggered",

stop: "[11] Thresholds Stopped",

state: "[12] Switch State",

state_val: {

0: "Off",

1: "On",

},

state_noval: "Unknown switch state (@)",

lat: "[13] Latitude",

lon: "[14] Longitude",

info: "[15] Firmware Info",

ccrc: "[16] Config CRC",

cfrom: "[17] Config from position",

cto: "[18] Config to position",

config: "[19] Configuration",

confver: "[01] Config layout version",

conflen: "[02] Config in use length",

deveui: "[03] LoRaWAN DevEUI",

appeui: "[04] LoRaWAN AppEUI",

region: "[05] LoRaWAN Region",

region_val: {

0: "EU868 (Europe 863-870 MHz ISM)",

1: "US915 (USA, Canada and South America 902-928 MHz ISM)",

2: "AU915 (Australia 915-928 MHz ISM)",

3: "AS923-1 (Asia and South America 923 MHz ISM)",

4: "IN866 (India 865-867 MHz ISM)",

```
  5: "AS923jp (Japan 923 MHz ISM with listen-before-talk (LBT) rules)",
  6: "KR920 (Korea 920-923 MHz ISM)",
  7: "AS923-2 (Indonesia and Vietnam 921 MHz ISM)",
  8: "AS923-3 (Africa, Middle East, Europe, Russia, Cuba and Philippines 916 MHz ISM)",
  9: "AS923-4 (Israel 917-920 MHz ISM)",
  255: "No LoRa (no LoRaWAN communication)",
},
region_noval: "Unknown region (@)",
adr: "[06] LoRaWAN ADR",
adr_val: {
  0: "Off",
  1: "On",
},
adr_noval: "Unknown ADR value (@)",
uport: "[07] LoRaWAN Upload Port",
hbport: "[08] LoRaWAN Upload Port for heartbeat/reminder",
confrm: "[09] LoRaWAN Confirm",
confrm_val: {
  0: "Off",
  1: "On",
},
confrm_noval: "Unknown confirm value (@)",
hbconfrm: "[10] LoRaWAN Confirm heartbeat/reminder",
hbconfrm_val: {
  0: "Off",
  1: "On",
},
hbconfrm_noval: "Unknown confirm value (@)",
dutycycl: "[11] LoRaWAN Duty cycle",
dutycycl_val: {
  0: "Always",
  1: "Not for new alarm/switch",
```

```
  2: "Not for alarm/switch and retries",
},
dutycycl_noval: "Unknown duty cycle value (@)",
unittype: "[12] LoRaButton Unit type",
unittype_val: {
  0: "Alarm Unit",
  1: "Vote Unit",
  2: "Switch Unit",
},
unittype_noval: "Unknown unit type value (@)",
numbut: "[13] LoRaButton Number of buttons",
butitv: "[14] LoRaButton Button press and switch interval [sec]",
buztog: "[15] LoRaButton Button / switch on buzzer feedback toggle interval [sec]",
buzdur: "[16] LoRaButton Button / switch on buzzer feedback duration [sec]",
ledtog: "[17] LoRaButton Button / switch on LED feedback toggle interval [sec]",
leddur: "[18] LoRaButton Button / switch on LED feedback duration [sec]",
alretr: "[19] LoRaButton Alarm / switch on retries",
alretitv: "[20] LoRaButton Alarm / switch retry interval [sec]",
butpin: "[21] LoRaButton Button $ pin",
butpin_val: {
  255: "No pin",
},
ledpin: "[22] LoRaButton Button $ LED pin",
ledpin_val: {
  255: "No pin",
  128: "Mix LED 1",
  129: "Mix LED 2",
  130: "Mix LED 3",
  131: "Mix LED 4",
},
voteaitv: "[23] LoRaButton Vote accumulation interval [sec]",
voteathr: "[24] LoRaButton Vote accumulation threshold per button",
```

```
mtitv: "[25] LoRaButton Heartbeat/measurement/reminder interval base [sec]",
stdel: "[26] LoRaButton Transmit first heartbeat/measurement/reminder [sec]",
amtitv: "[27] LoRaButton Heartbeat/measurement/reminder interval extra [sec]",
senspow: "[28] LoRaButton Sensors power pin",
senspow_val: {
  255: "No pin",
},
tcitv: "[29] LoRaButton Sensor Threshold check interval [sec]",
tmeas: "[30] LoRaButton Sensor Threshold $ measurement",
tmeas_val: {
  0: "Temperature [0.1 K]",
  1: "Relative Humidity [%]",
  4: "CO₂ [ppm]",
  6: "Distance [mm]",
  8: "Digital input $",
  254: "Voltage [10 mV]",
  255: "None/disabled",
},
tmeas_noval: "Unknown Thresholds Measurement value (@)",
ttval: "[31] LoRaButton Sensor Threshold $ trigger value",
tsval: "[32] LoRaButton Sensor Threshold $ stop value",
tbuztog: "[33] LoRaButton Sensor Feedback $ buzzer toggle interval [sec]",
tbuzdur: "[34] LoRaButton Sensor Feedback $ buzzer duration [sec]",
tbuzrem: "[35] LoRaButton Sensor Feedback $ buzzer reminder interval [sec]",
tledpin: "[36] LoRaButton Sensor Feedback $ LED pin",
tledpin_val: {
  255: "No pin",
  128: "Mix LED 1",
  129: "Mix LED 2",
  130: "Mix LED 3",
  131: "Mix LED 4",
},
```

```
tledtog: "[37] LoRaButton Sensor Feedback $ LED toggle interval [sec]",
tleddur: "[38] LoRaButton Sensor Feedback $ LED duration [sec]",
tledrem: "[39] LoRaButton Sensor Feedback $ LED reminder interval [sec]",
subband: "[40] LoRaWAN Sub Band for US like regions (1-8)",
subband_val: {
  0: "No subband direction",
},
scd30tmp: "[41] LoRaButton Sensor SCD30 Temperature Offset [°C]",
scd30alt: "[42] LoRaButton Sensor SCD30 Altitude [m]",
power: "[43] LoRaButton Power",
power_val: {
  0: "External Power",
  1: "2 x AA Alkaline",
  2: "2 x AA Lithium 1.5 Volt",
},
ofbuztog: "[44] LoRaButton Switch off buzzer feedback toggle interval [sec]",
ofbuzdur: "[45] LoRaButton Switch off buzzer feedback duration [sec]",
onledpin: "[46] LoRaButton Switch on LED pin",
onledpin_val: {
  255: "No pin",
  128: "Mix LED 1",
  129: "Mix LED 2",
  130: "Mix LED 3",
  131: "Mix LED 4",
},
ofledpin: "[47] LoRaButton Switch off LED pin",
ofledpin_val: {
  255: "No pin",
  128: "Mix LED 1",
  129: "Mix LED 2",
  130: "Mix LED 3",
  131: "Mix LED 4",
```

```
},
ofledtog: "[48] LoRaButton Switch off LED feedback toggle interval [sec]",
ofleddur: "[49] LoRaButton Switch off LED feedback duration [sec]",
ofretr: "[50] LoRaButton Switch off retries",
ncbuztog: "[51] LoRaButton Alarm No Confirmation buzzer feedback toggle interval [sec]",
ncbuzdur: "[52] LoRaButton Alarm No Confirmation buzzer feedback duration [sec]",
ncledpin: "[53] LoRaButton Alarm No Confirmation LED pin",
ncledpin_val: {
  255: "No pin",
  128: "Mix LED 1",
  129: "Mix LED 2",
  130: "Mix LED 3",
  131: "Mix LED 4",
},
ncledtog: "[54] LoRaButton Alarm No Confirmation LED feedback toggle interval [sec]",
ncleddur: "[55] LoRaButton Alarm No Confirmation LED feedback duration [sec]",
cbuztog: "[56] LoRaButton Alarm Confirmation buzzer feedback toggle interval [sec]",
cbuzdur: "[57] LoRaButton Alarm Confirmation buzzer feedback duration [sec]",
cledpin: "[58] LoRaButton Alarm Confirmation LED pin",
cledpin_val: {
  255: "No pin",
  128: "Mix LED 1",
  129: "Mix LED 2",
  130: "Mix LED 3",
  131: "Mix LED 4",
},
cledtog: "[59] LoRaButton Alarm Confirmation LED feedback toggle interval [sec]",
cleddur: "[60] LoRaButton Alarm Confirmation LED feedback duration [sec]",
choldoff: "[61] LoRaButton Alarm Confirmation feedback holdoff [sec]",
mixledpin: "[62] LoRaButton Mix LED $ LED $ pin",
mixledpin_val: {
  255: "No pin",
```

```
},
mixledlev: "[63] LoRaButton Mix LED $ LED $ level [%]",
diginp: "[64] Digital Input $ pin",
ttameas: "[65] LoRaButton Sensor Threshold $ trigger additional measurements",
tsameas: "[66] LoRaButton Sensor Threshold $ stop additional measurements",
uidtime: "[67] LoRaButton UID read timeout [sec]",
uidtime_val: {
 0: "UID reader disabled",
},
uidbcycl: "[68] LoRaButton UID read success beep [cycles]",
uidbcycl_val: {
 0: "No beep",
},
uidsucc: "[69] LoRaButton UID read success action",
uidsucc_val: {
 0: "Send Alarm and immediately give confirm feedback (when configured)",
 1: "Send Alarm and (when configured) wait for confirm downlink",
},
uidsucc_noval: "Unknown UID read success action value (@)",
uidfail: "[70] LoRaButton UID read failure action",
uidfail_val: {
 0: "Don't send Alarm and immediately give non confirm feedback (when configured)",
 1: "Send Alarm and (when configured) wait for confirm downlink",
 2: "Send Alarm and immediately give non confirm feedback (when configured)",
},
uidfail_noval: "Unknown UID read failure action value (@)",
uidbmask: "[71] LoRaButton UID read button mask (disable UID read for indicated buttons)",
buzbmask: "[72] LoRaButton Buzzer button mask (disable buzzer for indicated buttons, both button feedback and alarm (non) confirm)",
result: "[20] Update Result",
result_val: {
 0: "Success",
```

```
  1: "Incomplete (update length exceeds payload)",
  2: "Failed (not writable or position error)",
  3: "No length field",
  4: "Failed (max heartbeat/measurement/reminder interval increase more than double)",
},
result_noval: "Unknown update result (@)",
succcnt: "[21] Update Success count",
scd30result: "[22] SCD30 Result",
scd30result_val: {
  0: "Success",
  1: "Start Timeout",
  2: "Address Timeout",
  3: "Data Timeout",
  7: "Stop Timeout",
  32: "Address NACK",
  48: "Data NACK",
  56: "Arbitration lost",
},
alarmid: "[23] Alarm ID",
feedbacktime: "[24] Feedback Time [min]",
feedbacks: "[25] Feedbacks",
uiderror: "[26] UID Error",
uiderror_val: {
  32: "RFID/NFC Reader not detected at startup",
  48: "No tag/card found",
  64: "Lost communication with RFID/NFC Reader",
  96: "Reading card/tag failed, position it closer",
  112: "Multiple cards/tags, present only one",
  144: "Card/tag not yet supported",
  160: "Reading card/tag interrupted",
  176: "UID too long, maximal 10 bytes",
  192: "Tag/card with Random UID, not supported",
```

},

uidtype: "[27] UID Type",

uidtype_val: {

0: "Generic passive 106 kbps (ISO/IEC14443-4A, Mifare and DEP)",

1: "Generic passive 212 kbps (FeliCa and DEP)",

2: "Generic passive 424 kbps (FeliCa and DEP)",

3: "Passive 106 kbps ISO/IEC14443-4B",

4: "Innovision Jewel/Topaz tag",

16: "Mifare card",

17: "FeliCa 212 kbps card",

18: "FeliCa 424 kbps card",

32: "Passive 106 kbps ISO/IEC14443-4A",

35: "Passive 106 kbps ISO/IEC14443-4B",

64: "DEP passive 106 kbps",

65: "DEP passive 212 kbps",

66: "DEP passive 424 kbps",

128: "DEP active 106 kbps",

129: "DEP active 212 kbps",

130: "DEP active 424 kbps",

},

uid: "[28] UID",

left: "[88] Undecoded Payload Left [hex]",

error: "[89] Decode error",

error_val: {

0: "Empty",

10: "Unexpected end, no (complete) voltage",

20: "Unexpected end, no OpCode",

30: "Unknown OpCode",

35: "Unexpected end, UID Error requires Error value",

40: "Unexpected end, OpCode Votes requires #Buttons",

50: "Too many buttons, maximum is 12",

60: "Unexpected end, incomplete Button Totals",

```
    70: "Unexpected end, incomplete Button Counts",
    80: "Unexpected end, OpCode Measurements/Thresholds requires Measurements",
    90: "Unexpected end, measurements without (complete) Temperature value",
    100: "Unexpected end, measurements without Relative Humidity value",
    110: "Unexpected end, measurements without (complete) Pressure value",
    120: "Unexpected end, measurements without (complete) Light value",
    130: "Unexpected end, measurements without (complete) CO₂ value",
    140: "Unexpected end, measurements without (complete) TVOC value",
    143: "Unexpected end, measurements without (complete) Distance value",
    144: "Unexpected end, measurements without more Measurements",
    145: "Unexpected end, measurements without Digital Inputs",
    150: "Unexpected end, OpCode Thresholds requires Threshold info",
    160: "Unexpected end, OpCode Location requires (complete) Latitude",
    170: "Unexpected end, OpCode Location requires (complete) Longitude",
    175: "Unexpected end, OpCode Switch/Reminder requires Switch State",
    177: "Unexpected end, OpCode Feedback requires Feedback Time",
    180: "Unexpected end, OpCode Show/Update require (complete) CRC",
    190: "Unexpected end, OpCode Show requires Config Start",
    200: "Unexpected end, OpCode Show requires Config Length",
    210: "Unexpected end, OpCode Show without (complete) Config Content",
    220: "Unexpected end, OpCode Update requires Update Result",
    230: "Unknown Update Result",
    240: "Unexpected end, OpCode SCD30 requires SCD30 Result",
  },
  hex: "[99] Complete payload [hex]",
};

function Descriptions(decode) {
  if (decode === null || typeof decode !== "object" || Array.isArray(decode)) return decode;
  var descriptions = {};
  for (var key in decode) {
    var value = Descriptions(decode[key]);
```

```
    var labels = key.split("_");
    var description = translation[labels[0]];
    if (fahrenheit && labels[0] == "temp" && translation["temp_f"]) description =
translation["temp_f"];
    if (typeof description === "string") {
      for (var label_d = 1; label_d < labels.length; label_d += 1) {
        description = description.replace("$", labels[label_d]);
      }
      key = description;
    }
    var value_descriptions = translation[labels[0] + "_val"];
    if (value_descriptions !== undefined && value_descriptions !== null) {
      var value_description = value_descriptions[value];
      if (value_description === undefined) {
        var value_nodescription = translation[labels[0] + "_noval"];
        if (typeof value_nodescription === "string") value = value_nodescription.replace("@", value);
      } else {
        value = value_description;
      }
    }
    if (typeof value === "string") {
      for (var label_v = 1; label_v < labels.length; label_v += 1) {
        value = value.replace("$", labels[label_v]);
      }
    }
    descriptions[key] = value;
  }
  return descriptions;
}

function MutelcorLoRaButtonDecode(bytes, port) {
  var decoded = {};
```

```
var pos = 0;
decoded.hex = toHexMSBF(bytes);
decoded.port = port;
if (bytes.length === 0) {
  decoded.error = 0;
  return addPayloadLeft(decoded, bytes, pos);
}
decoded.version = bytes[pos++];
if (bytes.length < pos + 2) {
  decoded.error = 10;
  return addPayloadLeft(decoded, bytes, pos);
}
decoded.voltage = (bytes[pos++] * 256 + bytes[pos++]) / 100;
if (bytes.length < pos + 1) {
  decoded.error = 20;
  return addPayloadLeft(decoded, bytes, pos);
}
var opcode = bytes[pos++];
decoded.opcode = opcode;
if (!(opcode in translation.opcode_val)) decoded.error = 30;
if (decoded.opcode === 1) {
  if (pos < bytes.length && pos + 2 != bytes.length) {
    decoded.buttons = ["1", "2", "3", "4", "5", "6", "7", "8"].filter(bitmaskFilter, bytes[pos++]);
  }
  if (pos + 1 < bytes.length) {
    decoded.alarmid = bytes[pos++] * 256 + bytes[pos++];
  }
  if (pos < bytes.length) {
    var uidtype = bytes[pos++];
    if (uidtype === 255) {
        if (pos < bytes.length) {
          decoded.uiderror = bytes[pos++];
```

```
      } else {
        decoded.error = 35;
        return addPayloadLeft(decoded, bytes, pos);
      }
    } else {
      decoded.uidtype = uidtype;
      decoded.uid = (uid_type_prefix ? toHex(uidtype) : "") + toHexMSBF(bytes.slice(pos));
      pos = bytes.length;
    }
  }
}
if (decoded.opcode === 2 && bytes.length < pos + 1) {
  decoded.error = 40;
  return addPayloadLeft(decoded, bytes, pos);
}
if (decoded.opcode === 2 || decoded.opcode === 0 && pos + 1 <= bytes.length) {
  var buttons = bytes[pos++];
  decoded.buttons = buttons;
  if (12 < buttons) {
    decoded.error = 50;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.totals = {};
  for (var button_t = 1; button_t <= buttons; button_t += 1) {
    if (bytes.length < pos + 2) {
      decoded.error = 60;
      return addPayloadLeft(decoded, bytes, pos);
    }
    if (pos + 2 <= bytes.length) {
      decoded.totals[button_t] = bytes[pos++] * 256 + bytes[pos++];
    }
  }
```

```
  if (decoded.opcode === 2) {
    decoded.counts = {};
    for (var button_c = 1; button_c <= buttons; button_c += 1) {
      if (bytes.length < pos + 1) {
        decoded.error = 70;
        return addPayloadLeft(decoded, bytes, pos);
      }
      decoded.counts[button_c] = bytes[pos++];
    }
    if (pos + 4 <= bytes.length) {
      decoded.qcrc = ((bytes[pos++] * 256 + bytes[pos++]) * 256 + bytes[pos++]) * 256 + bytes[pos++];
    }
  }
}
if (decoded.opcode === 3 || decoded.opcode === 5) {
  if (bytes.length < pos + 1) {
    decoded.error = 80;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas = {};
  var measurements = bytes[pos++];
  if (measurements & 1) {
    if (bytes.length < pos + 2) {
      decoded.error = 90;
      return addPayloadLeft(decoded, bytes, pos);
    }
    decoded.meas.temp = (bytes[pos++] * 256 + bytes[pos++]) / 10;
    if (fahrenheit) decoded.meas.temp = Math.round((decoded.meas.temp * 1.8 + 32) * 100) / 100;
  }
  if (measurements & 2) {
    if (bytes.length < pos + 1) {
      decoded.error = 100;
```

```
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas.rh = bytes[pos++];
}
if (measurements & 4) {
  if (bytes.length < pos + 2) {
    decoded.error = 110;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas.press = (bytes[pos++] * 256 + bytes[pos++]) / 10;
}
if (measurements & 8) {
  if (bytes.length < pos + 2) {
    decoded.error = 120;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas.light = bytes[pos++] * 256 + bytes[pos++];
}
if (measurements & 16) {
  if (bytes.length < pos + 2) {
    decoded.error = 130;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas.co2 = bytes[pos++] * 256 + bytes[pos++];
}
if (measurements & 32) {
  if (bytes.length < pos + 2) {
    decoded.error = 140;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas.tvoc = bytes[pos++] * 256 + bytes[pos++];
}
```

```
if (measurements & 64) {
  if (bytes.length < pos + 2) {
    decoded.error = 143;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.meas.dist = bytes[pos++] * 256 + bytes[pos++];
}
if (measurements & 128) {
  if (bytes.length < pos + 2) {
    decoded.error = 144;
    return addPayloadLeft(decoded, bytes, pos);
  }
  measurements = bytes[pos++];
  if (measurements & 1) {
    if (bytes.length < pos + 2) {
      decoded.error = 145;
      return addPayloadLeft(decoded, bytes, pos);
    }
    var digital_inputs = bytes[pos++];
    var dinputs = {};
    for (var diginp = 0; diginp < 4; diginp += 1) {
      if (digital_inputs & (1 << diginp)) dinputs[diginp + 1] = (digital_inputs & (1 << (diginp + 4))) != 0;
    }
    decoded.meas.dinputs = dinputs;
  }
}
if (decoded.opcode === 5) {
  if (bytes.length < pos + 1) {
    decoded.error = 150;
    return addPayloadLeft(decoded, bytes, pos);
  }
  var threshold_info = bytes[pos++];
```

```
    var triggered = ["1", "2", "3", "4"].filter(bitmaskFilter, threshold_info & 0x0F);
    if (triggered && 0 < triggered.length) decoded.trigger = triggered;
    var stopped = ["1", "2", "3", "4"].filter(bitmaskFilter, threshold_info >> 4);
    if (stopped && 0 < stopped.length) decoded.stop = stopped;
  }
  if (pos + 1 <= bytes.length) {
    decoded.state = bytes[pos++];
  }
}
if (decoded.opcode === 4) {
  if (bytes.length < pos + 4) {
    decoded.error = 160;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.lat = ((bytes[pos++] * 256 + bytes[pos++]) * 256 + bytes[pos++]) * 256 + bytes[pos++];
  if (decoded.lat > 2147483647) decoded.lat -= 4294967296;
  decoded.lat /= 10000000;
  if (bytes.length < pos + 4) {
    decoded.error = 170;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.lon = ((bytes[pos++] * 256 + bytes[pos++]) * 256 + bytes[pos++]) * 256 + bytes[pos++];
  if (decoded.lon > 2147483647) decoded.lon -= 4294967296;
  decoded.lon /= 10000000;
}
if (decoded.opcode === 6 || decoded.opcode === 7) {
  if (bytes.length < pos + 1) {
    decoded.error = 175;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.state = bytes[pos++];
}
```

```
if (decoded.opcode === 80) {
  if (bytes.length < pos + 1) {
    decoded.error = 177;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.feedbacktime = bytes[pos++];
  if (pos + 1 <= bytes.length) {
    var feedbacks_bitmask = bytes[pos++];
    var feedbacks = ["1", "2", "3", "4"].filter(bitmaskFilter, feedbacks_bitmask);
    if (feedbacks && 0 < feedbacks.length) decoded.feedbacks = feedbacks;
  }
}
if (decoded.opcode === 112) {
  var info = "";
  while (pos < bytes.length) {
    info += String.fromCharCode(bytes[pos++]);
  }
  decoded.info = info
}
if (decoded.opcode === 113 || decoded.opcode === 114) {
  if (bytes.length < pos + 2) {
    decoded.error = 180;
    return addPayloadLeft(decoded, bytes, pos);
  }
  decoded.ccrc = bytes[pos++] * 256 + bytes[pos++];
  if (decoded.opcode === 113) {
    if (bytes.length < pos + 1) {
      decoded.error = 190;
      return addPayloadLeft(decoded, bytes, pos);
    }
    var config_start = bytes[pos++];
    decoded.cfrom = config_start;
```

```
if (bytes.length < pos + 1) {
  decoded.error = 200;
  return addPayloadLeft(decoded, bytes, pos);
}
var config_length = bytes[pos++];
var config_index = config_start;
var config_end = config_start + config_length
if (bytes.length < pos + config_length) config_end = config_start + bytes.length - pos;
decoded.cto = config_end - 1;
decoded.config = {};
while (config_index < config_end) {
  var field = null;
  var value = bytes[pos];
  var shift = 1;
  var config_pos = config_index;
  if (51 <= config_index && config_index <= 62) config_pos = 51;
  if (63 <= config_index && config_index <= 74) config_pos = 63;
  if (82 <= config_index && config_index <= 85) config_pos = 82;
  if (86 <= config_index && config_index <= 93) config_pos = 86;
  if (94 <= config_index && config_index <= 101) config_pos = 94;
  if (102 <= config_index && config_index <= 105) config_pos = 102;
  if (106 <= config_index && config_index <= 109) config_pos = 106;
  if (110 <= config_index && config_index <= 113) config_pos = 110;
  if (114 <= config_index && config_index <= 117) config_pos = 114;
  if (118 <= config_index && config_index <= 121) config_pos = 118;
  if (122 <= config_index && config_index <= 125) config_pos = 122;
  if (126 <= config_index && config_index <= 129) config_pos = 126;
  if (152 <= config_index && config_index <= 163) config_pos = 152;
  if (164 <= config_index && config_index <= 175) config_pos = 164;
  if (176 <= config_index && config_index <= 179) config_pos = 176;
  if (180 <= config_index && config_index <= 183) config_pos = 180;
  if (184 <= config_index && config_index <= 187) config_pos = 184;
```

```
var index = config_index - config_pos;
switch (config_pos) {
  case 0:
    field = "confver";
    break;
  case 1:
    if (config_index + 2 <= config_end) {
      field = "conflen";
      value += bytes[pos + 1] * 256;
      shift = 2;
    }
    break;
  case 3:
    if (config_index + 8 <= config_end) {
      field = "deveui";
      value = toHexLSBF(bytes.slice(pos), 8);
      shift = 8;
    }
    break;
  case 11:
    if (config_index + 8 <= config_end) {
      field = "appeui";
      value = toHexLSBF(bytes.slice(pos), 8);
      shift = 8;
    }
    break;
  case 35:
    field = "region";
    break;
  case 36:
    field = "adr";
    break;
```

```
case 37:
  field = "uport";
  break;
case 38:
  field = "hbport";
  break;
case 39:
  field = "confrm";
  break;
case 40:
  field = "hbconfrm";
  break;
case 41:
  field = "dutycycl";
  break;
case 42:
  field = "unittype";
  break;
case 43:
  field = "numbut";
  break;
case 44:
  field = "butitv"
  value /= 4;
  break;
case 45:
  field = "buztog";
  value /= 4;
  break;
case 46:
  field = "buzdur";
  value /= 4;
```

```
  break;
case 47:
  field = "ledtog"
  value /= 4;
  break;
case 48:
  field = "leddur";
  value /= 4;
  break;
case 49:
  field = "alretr";
  break;
case 50:
  field = "alretitv";
  break;
case 51:
  field = "butpin_" + ("0" + (index + 1)).slice(-2);
  break;
case 63:
  field = "ledpin_" + ("0" + (index + 1)).slice(-2);
  break;
case 75:
  field = "voteaitv";
  value *= 15;
  break;
case 76:
  field = "voteathr";
  break;
case 77:
  field = "mtitv";
  value *= 360;
  break;
```

```
case 78:
  field = "stdel";
  value *= 15;
  break;
case 79:
  field = "amtitv";
  value *= 2;
  break;
case 80:
  field = "senspow";
  break;
case 81:
  field = "tcitv";
  value /= 4;
  break;
case 82:
  field = "tmeas_" + (index + 1);
  break;
case 86:
  if (config_index + 2 <= config_end) {
    field = "ttval_" + ((index >> 1) + 1);
    value += bytes[pos + 1] * 256;
    shift = 2;
  }
  break;
case 94:
  if (config_index + 2 <= config_end) {
    field = "tsval_" + ((index >> 1) + 1);
    value += bytes[pos + 1] * 256;
    shift = 2;
  }
  break;
```

```
case 102:
  field = "tbuztog_" + (index + 1);
  value /= 4;
  break;
case 106:
  field = "tbuzdur_" + (index + 1);
  value /= 4;
  break;
case 110:
  field = "tbuzrem_" + (index + 1);
  value *= 60;
  break;
case 114:
  field = "tledpin_" + (index + 1);
  break;
case 118:
  field = "tledtog_" + (index + 1);
  value /= 4;
  break;
case 122:
  field = "tleddur_" + (index + 1);
  value /= 4;
  break;
case 126:
  field = "tledrem_" + (index + 1);
  value *= 60;
  break;
case 130:
  field = "subband";
  break;
case 131:
  field = "scd30tmp";
```

```
  value /= 10;
  break;
case 132:
  field = "scd30alt";
  value *= 10;
  break;
case 133:
  field = "power";
  break;
case 134:
  field = "ofbuztog";
  value /= 4;
  break;
case 135:
  field = "ofbuzdur";
  value /= 4;
  break;
case 136:
  field = "onledpin";
  break;
case 137:
  field = "ofledpin";
  break;
case 138:
  field = "ofledtog";
  value /= 4;
  break;
case 139:
  field = "ofleddur";
  value /= 4;
  break;
case 140:
```

```
  field = "ofretr";
  break;
case 141:
  field = "ncbuztog";
  value /= 16;
  break;
case 142:
  field = "ncbuzdur";
  value /= 4;
  break;
case 143:
  field = "ncledpin";
  break;
case 144:
  field = "ncledtog";
  value /= 16;
  break;
case 145:
  field = "ncleddur";
  value /= 4;
  break;
case 146:
  field = "cbuztog";
  value /= 16;
  break;
case 147:
  field = "cbuzdur";
  value /= 4;
  break;
case 148:
  field = "cledpin";
  break;
```

```
case 149:
  field = "cledtog";
  value /= 16;
  break;
case 150:
  field = "cleddur";
  value /= 4;
  break;
case 151:
  field = "choldoff";
  value /= 4;
  break;
case 152:
  field = "mixledpin_" + (index % 4 + 1) + "_" + (Math.floor(index / 4) + 1);
  break;
case 164:
  field = "mixledlev_" + (index % 4 + 1) + "_" + (Math.floor(index / 4) + 1);
  value = Math.round((value * 1000) / 255) / 10;
  break;
case 176:
  field = "diginp_" + (index + 1);
  break;
case 180:
  field = "ttameas_" + (index + 1);
  break;
case 184:
  field = "tsameas_" + (index + 1);
  break;
case 188:
  field = "uidtime";
  value /= 4;
  break;
```

```
      case 189:
        field = "uidbcycl";
        break;
      case 190:
        field = "uidsucc";
        break;
      case 191:
        field = "uidfail";
        break;
      case 192:
        field = "uidbmask";
        value = ["1", "2", "3", "4", "5", "6", "7", "8"].filter(bitmaskFilter, value);
        break;
      case 193:
        field = "buzbmask";
        value = ["1", "2", "3", "4", "5", "6", "7", "8"].filter(bitmaskFilter, value);
        break;
    }
    if (field !== null) decoded.config[field] = value;
    pos += shift;
    config_index += shift;
  }
  if (config_index < config_start + config_length) {
    decoded.error = 210;
    return addPayloadLeft(decoded, bytes, pos);
  }
}
if (decoded.opcode === 114) {
  if (bytes.length < pos + 1) {
    decoded.error = 220;
    return addPayloadLeft(decoded, bytes, pos);
  }
```

```
    var result = bytes[pos++];
    decoded.result = result;
    if (!(result in translation.result_val)) decoded.error = 230;
    // Success count was added in version 1.4.1, to be backwards compatible we don't give an error when omitted
    if (pos < bytes.length) {
      decoded.succcnt = bytes[pos++];
    }
   }
  }
  if (decoded.opcode === 128) {
   if (bytes.length < pos + 1) {
    decoded.error = 240;
    return addPayloadLeft(decoded, bytes, pos);
   }
   decoded.scd30result = bytes[pos++];
  }
  return addPayloadLeft(decoded, bytes, pos);
}

function addPayloadLeft(decoded, bytes, pos) {
  if (pos < bytes.length) decoded.left = toHexMSBF(bytes.slice(pos));
  return decoded;
}

function bitmaskFilter(value, pos) {
  return this & (1 << pos);
}

function toHex(byte) {
  return ("0" + byte.toString(16).toUpperCase()).slice(-2);
}
```

```
function toHexMSBF(buf, len) {
  if (len === undefined) len = buf.length;
  var output = "";
  for (var i = 0; i < len; i += 1) output += toHex(buf[i]);
  return output;
}

function toHexLSBF(buf, len) {
  if (len === undefined) len = buf.length;
  var output = "";
  for (var i = len - 1; 0 <= i; i -= 1) output += toHex(buf[i]);
  return output;
}
```