Honeywell

LoRaWAN Payload Decoder

Note: The below decoder is a sample tested for ExperionEHM using LoRaWAN providers such as Chirpstack and The Things Network. The decoder varies depending on the network server that you chose for your needs.

```
function decodeUplink(input) {
    var data = input.bytes;
    var valid = true;

    if (typeof Decoder === "function") {
        data = Decoder(data, input.fPort);
    }

    if (typeof Converter === "function") {
        data = Converter(data, input.fPort);
    }

    if (typeof Validator === "function") {
        valid = Validator(data, input.fPort);
    }

    if (valid) {
        return {
            data: data
        };
    } else {
        return {
            data: {},
            errors: ["Invalid data received"]
        };
    }
```

```
}

function Decoder(byte, port)
{
   const EVENT_PROPERTY_NAMES =
["Ambient_Temp","Surface_Temp","Ambient_Pressure","Ambient_Humidity","Battery","Vib_Accel_X_A
xis","Vib_Accel_Y_Axis","Vib_Accel_Z_Axis","Vib_Velocity_X_Axis","Vib_Velocity_Y_Axis","Vib_Velocity_
Z_Axis","Audio"];

   var pktTime = (byte[3] << 24) | (byte[2] << 16) | (byte[1] << 8) | byte[0];

   // Periodic measurements
   if(port == 0x02)
   {
      const GCONVERSION = 9.8;

      for(let i = 4; i < 10; i++){
         if(byte[i] > 127){
            byte[i] = (256 - byte[i])*(-1);
         }
      }

      var Ambient_Temp_Max = (byte[4]);
      var Ambient_Temp_Min = (byte[5]);
      var Ambient_Temp_Avg = (byte[6]);
      var Surface_Temp_Max = (byte[7]);
      var Surface_Temp_Min = (byte[8]);
      var Surface_Temp_Avg = (byte[9]);
      var Ambient_Pressure_Max = (((byte[10] * 3) + 300));
      var Ambient_Pressure_Min = (((byte[11] * 3) + 300));
      var Ambient_Pressure_Avg = (((byte[12] * 3) + 300));
```

```
var Ambient_Humidity_Max = (byte[13]);
var Ambient_Humidity_Min = (byte[14]);
var Ambient_Humidity_Avg = (byte[15]);

var Vib_Accel_X_Axis_Max = (byte[16]) /GCONVERSION;
var Vib_Accel_X_Axis_Min = (byte[17]) /GCONVERSION;
var Vib_Accel_X_Axis_RMS = (byte[18]) /GCONVERSION;

var Vib_Accel_Y_Axis_Max = (byte[19]) /GCONVERSION;
var Vib_Accel_Y_Axis_Min = (byte[20]) /GCONVERSION;
var Vib_Accel_Y_Axis_RMS = (byte[21]) /GCONVERSION;

var Vib_Accel_Z_Axis_Max = (byte[22]) /GCONVERSION;
var Vib_Accel_Z_Axis_Min = (byte[23]) /GCONVERSION;
var Vib_Accel_Z_Axis_RMS = (byte[24]) /GCONVERSION;

var Vib_Velocity_X_Axis_Max = (byte[25])/10; // mm/sec with factor 10 from device
var Vib_Velocity_X_Axis_Min = (byte[26])/10;
var Vib_Velocity_X_Axis_RMS = (byte[27])/10;
var Vib_Velocity_Y_Axis_Max = (byte[28])/10;
var Vib_Velocity_Y_Axis_Min = (byte[29])/10;
var Vib_Velocity_Y_Axis_RMS = (byte[30])/10;
var Vib_Velocity_Z_Axis_Max = (byte[31])/10;
var Vib_Velocity_Z_Axis_Min = (byte[32])/10;
var Vib_Velocity_Z_Axis_RMS = (byte[33])/10;
var Audio_dBSPL = (byte[34]);
var Audio_Max = (byte[35]);
var Audio_Min = (byte[36]);
var Remaining_battery_perc = byte[37];
```

```
var decoded =
{
   timestamp : pktTime,
   var_Ambient_Temp_Max:Ambient_Temp_Max,
   var_Ambient_Temp_Min:Ambient_Temp_Min,
   var_Ambient_Temp_Avg:Ambient_Temp_Avg,
   var_Surface_Temp_Max:Surface_Temp_Max,
   var_Surface_Temp_Min:Surface_Temp_Min,
   var_Surface_Temp_Avg:Surface_Temp_Avg,
   var_Ambient_Pressure_Max:Ambient_Pressure_Max,
   var_Ambient_Pressure_Min:Ambient_Pressure_Min,
   var_Ambient_Pressure_Avg:Ambient_Pressure_Avg,
   var_Ambient_Humidity_Max:Ambient_Humidity_Max,
   var_Ambient_Humidity_Min:Ambient_Humidity_Min,
   var_Ambient_Humidity_Avg:Ambient_Humidity_Avg,
   var_Vib_Accel_X_Axis_Max:Vib_Accel_X_Axis_Max,
   var_Vib_Accel_X_Axis_Min:Vib_Accel_X_Axis_Min,
   var_Vib_Accel_X_Axis_RMS:Vib_Accel_X_Axis_RMS,
   var_Vib_Accel_Y_Axis_Max:Vib_Accel_Y_Axis_Max,
   var_Vib_Accel_Y_Axis_Min:Vib_Accel_Y_Axis_Min,
   var_Vib_Accel_Y_Axis_RMS:Vib_Accel_Y_Axis_RMS,
   var_Vib_Accel_Z_Axis_Max:Vib_Accel_Z_Axis_Max,
   var_Vib_Accel_Z_Axis_Min:Vib_Accel_Z_Axis_Min,
   var_Vib_Accel_Z_Axis_RMS:Vib_Accel_Z_Axis_RMS,
   var_Vib_Velocity_X_Axis_Max:Vib_Velocity_X_Axis_Max,
   var_Vib_Velocity_X_Axis_Min:Vib_Velocity_X_Axis_Min,
   var_Vib_Velocity_X_Axis_RMS:Vib_Velocity_X_Axis_RMS,
   var_Vib_Velocity_Y_Axis_Max:Vib_Velocity_Y_Axis_Max,
   var_Vib_Velocity_Y_Axis_Min:Vib_Velocity_Y_Axis_Min,
```

```
        var_Vib_Velocity_Y_Axis_RMS:Vib_Velocity_Y_Axis_RMS,
        var_Vib_Velocity_Z_Axis_Max:Vib_Velocity_Z_Axis_Max,
        var_Vib_Velocity_Z_Axis_Min:Vib_Velocity_Z_Axis_Min,
        var_Vib_Velocity_Z_Axis_RMS:Vib_Velocity_Z_Axis_RMS,
        var_Audio_Max:Audio_Max,
        var_Audio_Min:Audio_Min,
        var_Audio_dBSPL:Audio_dBSPL,
        var_Remaining_battery_perc:Remaining_battery_perc,
    };
    return decoded;
}

// Events
if(port == 8)
{
    var decoded_8 =
    {
        timestamp : pktTime,
    };

    var Sensor_Type = byte[4];
    var Event_Type = byte[5];

    if(Sensor_Type < 4)  // Regular Sensor measurement alarms
    {
        var Event_Data = byte[6];
                        if(Sensor_Type <= 1) // Ambient Temp, Surface Temp
                        {
                                if(Event_Data > 127)
                                {
```

```
                    Event_Data = (256 - byte[6])*(-1);
                }
            }
            else if(Sensor_Type == 2) // Ambient Pressure
            {
                Event_Data = ((byte[6] * 3) + 300);
            }

        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Type"] = Event_Type;
        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Data"] = Event_Data;
        return decoded_8;
    }
    else if( (Sensor_Type > 4) && (Sensor_Type <= 11) ) // Vibration & Acoustics alarms
    {
        var Freq_Band = byte[7];
        var Freq_Value = (byte[10] << 16) | (byte[9] << 8) | byte[8];
        var Amplitude = byte[11];

            if(Sensor_Type != 11) // acoustics doesn't have factor 10 for amplitude
                Amplitude = Amplitude/10;

        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Type"] = Event_Type;
        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Freq" + Freq_Band] =
Freq_Value;
        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Amp" + Freq_Band] =
Amplitude;
        return decoded_8;
    }
    else if(Sensor_Type == 4) // Battery Alarm
    {
        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Type"] = Event_Type;
```

```
        if(Event_Type == 1) // Battery Voltage Low
                                        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] +
"_Voltage"] = 3 + (byte[6] * 0.004);
                                else if(Event_Type == 2) // Battery Life Changed
                                        decoded_8["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] +
"_Life"] = byte[6];
        return decoded_8;
      }
   }


   // Diagnostics
   if(port == 11)
   {
      var Diag_Status = byte[5] << 8 | byte[4];
      var decoded_11 =
      {
         timestamp : pktTime,
         Diag_Status: Diag_Status,
      };


      // all measurements
      for(let Sensor_Type = 0; Sensor_Type < 12; Sensor_Type++) {
         decoded_11["event_" + EVENT_PROPERTY_NAMES[Sensor_Type] + "_Type"] = byte[Sensor_Type
+ 6];
      }


      return decoded_11;
   }
}
```