enLink LoRa Payload Structure

enLink packet structure is designed to be as efficient as possible. Data for multiple sensor values can be concatenated into a single message which can be easily decoded. If the LoRa payload length is restricted due to channel time limits, the whole message may be split into multiple "packets". These packets are simply multiple messages. Each message/packet will be split on a Sensor data boundary. This is done so messages are easily decoded, as each message will always have the first byte as a Data Type Identifier.

Payload structure

Sensor 1 Data (2 or more bytes)	Sensor 2 Data	Sensor *n* Data

Sensor Data

Sensor Data consists of a **Data Type Identifier** byte followed by the **Data Value** as one or more bytes. The number of bytes is determined by the Data Type Identifier and is fixed. See: Table 1: enLink Data Structure.

Example Message

XX	=	Data Type Identifier
XX	=	Data Byte Value

Payload Data	01	01	23	02	56	03	01	A4

Temperature [**0x01**] = ((0x01 * 256) + 0x23) / 10 = (256 + 35) / 10 = **29.1°C**

Humidity [**0x02**] = 0x56 = **86%RH**

Light Level [**0x03**] = (0x01 * 256) + 0xA4 = 256 + 164 = **420 Lux**

Each Data Type can use 1 or more bytes to send the value according to the following table.

Type Byte	Sensor	Sensor Range	Units	Num Bytes	Data Format	Data Range	Scaling
0x01	Temperature	-40 - 85	°C	2	S16	-3276.8 --> 3276.7°C	/ 10
0x02	Humidity	0 - 100	%	1	U8	0 --> 100 %RH	
0x03	Ambient Light	0.01 - 83k	lux	2	U16	0 --> 65535 lx	
0x04	Pressure	300 - 1100	mbar	2	U16	0 --> 65535 mbar	
0x05	Volatile Organic Compounds	0 - 500	IAQ	2	U16	0 --> 65535	
0x06	Oxygen	0 – 25.0	%	1	U8	0 --> 25.5%	/ 10
0x07	Carbon Monoxide	0 - 100	ppm	2	U16	0 --> 655.35 ppm	/ 100
0x08	Carbon Dioxide	0 - 2000	ppm	2	U16	0 --> 65535 ppm	
0x09	Ozone (O_3)	0 - 1	ppm	2	U16	0 --> 6.5535 ppm 0 --> 6553.5 ppb	/ 10000 / 10
0x0A	Air Pollutants: CO, Ammonia, Ethanol, H2, Methane / Propane / Iso-Butane.	Typically 100 – 1500	kΩ	2	U16	0 --> 6553.5 kΩ	/ 10
0x0B	Particulate Matter 2.5	0 - 1000	µg/m^3	2	U16	0 --> 65535 µg/m^3	
0x0C	Particulate Matter 10	0 - 1000	µg/m^3	2	U16	0 --> 65535 µg/m^3	
0x0D	Hydrogen Sulphide (H_2S)	0 - 100	ppm	2	U16	0 --> 655.35 ppm	/ 100
0x0E	Pulse ID + Pulse Counter		count	5	U32	Pulse ID: 0 --> 3; Value: 0 --> 2^32	
0x0F	Modbus Exception	MB Item + Exception Byte		2	U8	MB Item 0 --> 31 Value is Error Num	
0x10	Modbus Interval value as IEEE754 F32	MB Item + Interval Value as F32		5	F32	MB Item 0 --> 31 Value depends on source	
0x11	Modbus Cumulative value as IEEE754 F32	MB Item + Cumulative Value as F32		5	F32	MB Item 0 --> 31 Value depends on source	
0x12	bVOC – Breath VOC estimate equivalent		ppm	4	F32		
0x13	Detection count (PIR etc.)		count	4	U32	Value: 0 --> 4.2 billion	
0x14	Total occupied time (secs)		s	4	U32	Value: 0 --> 136 years	
0x15	Occupied Status		status	1	U8	0/1	
0x16	Liquid Level Status		status	1	U8	0/1	
0x17	Probe 1 Temperature	-55 to 125	°C	2	S16	-3276.8 --> 3276.7°C	/ 10
0x18	Probe 2 Temperature	-55 to 125	°C	2	S16	-3276.8 --> 3276.7°C	/ 10
0x19	Probe 3 Temperature	-55 to 125	°C	2	S16	-3276.8 --> 3276.7°C	/ 10
0x1A	Time temperature probe 1 has spent in 'in band' zone		s	4	U32	Value: 0 --> 136 years	
0x1B	Time temperature probe 2 has spent in 'in band' zone		s	4	U32	Value: 0 --> 136 years	
0x1C	Time temperature probe 3 has spent in 'in band' zone		s	4	U32	Value: 0 --> 136 years	
0x1D	Number of times in band alarm has been activated for temperature probe 1		count	2	U16	0 --> 65535	
0x1E	Number of times in band alarm has been activated for temperature probe 2		count	2	U16	0 --> 65535	
0x1F	Number of times in band alarm has been activated for temperature probe 3		count	2	U16	0 --> 65535	
0x20	Time temperature probe 1 has spent below low threshold		s	4	U32	Value: 0 --> 136 years	
0x21	Time temperature probe 2 has spent below low threshold		s	4	U32	Value: 0 --> 136 years	
0x22	Time temperature probe 3 has spent below low threshold		s	4	U32	Value: 0 --> 136 years	
0x23	Number of times low threshold alarm has been activated for temperature probe 1		count	2	U16	0 --> 65535	
0x24	Number of times low threshold alarm has been activated for temperature probe 2		count	2	U16	0 --> 65535	
0x25	Number of times low threshold alarm has been activated for temperature probe 3		count	2	U16	0 --> 65535	

Type Byte	Sensor	Sensor Range	Units	Num Bytes	Data Format	Data Range	Scaling
0x26	Time temperature probe 1 has spent above high threshold		s	4	U32	Value: 0 --> 136 years	
0x27	Time temperature probe 2 has spent above high threshold		s	4	U32	Value: 0 --> 136 years	
0x28	Time temperature probe 3 has spent above high threshold		s	4	U32	Value: 0 --> 136 years	
0x29	Number of times high threshold alarm has been activated for temperature probe 1		count	2	U16	0 --> 65535	
0x2A	Number of times high threshold alarm has been activated for temperature probe 2		count	2	U16	0 --> 65535	
0x2B	Number of times high threshold alarm has been activated for temperature probe 3		count	2	U16	0 --> 65535	
0x2C	Differential Pressure	+/- 5000	Pa	4	F32		
0x2D	Airflow	0 to 100	m/s	4	F32		
0x2E	Voltage	0 to 10 V	Volts	2	U16	0 - 65.535 V	/ 1000
0x2F	Current	0 to 20 mA	mA	2	U16	0 - 65.535 mA	/ 1000
0x30	Resistance	0 to 10 kOhm	Ohm	2	U16	0 - 65.535 kOhm	/ 1000
0x31	Leakage Detection (resistance rope)	Leak status changed	status	1	U8	0 or 1	
0x32	Vibration	Vibration status changed	status	1	U8	0 or 1	
0x3A	Pressure/Depth Transducer	0 to 50000+	mbar / mm	2	U16	0 -> 65535	
0x3B	Transducer Temperature	-40 - 85	°C	2	S16	-3276.8 --> 3276.7°C	/ 10
0x3F	CO_2e estimate equivalent		ppm	4	F32		
0x50	Sound Level Minimum		dB(A)	4	F32		
0x51	Sound Level Average		dB(A)	4	F32		
0x52	Sound Level Maximum		dB(A)	4	F32		
0x53	Nitric Oxide	0 - 100	ppm	2	U16	0 --> 655.35 ppm	/ 100
0x54	Nitrogen Dioxide	0 – 5	ppm	2	U16	0 --> 6.5535 ppm 0 --> 6553.5 ppb	/ 10000 / 10
0x55	Nitrogen Dioxide	0 – 20	ppm	2	U16	0 --> 65.535 ppm	/ 1000
0x56	Sulphur Dioxide	0 – 20	ppm	2	U16	0 --> 65.535 ppm	/ 1000
0x57	Particulate matter mass concentration at PM1.0		µg/m³	4	F32		
0x58	As above, PM2.5		µg/m³	4	F32		
0x59	As above, PM4.0		µg/m³	4	F32		
0x5A	As above, PM10.5		µg/m³	4	F32		
0x5B	Particulate matter number concentration at PM0.5		#/cm³	4	F32		
0x5C	As above, PM1.0		#/cm³	4	F32		
0x5D	As above, PM2.5		#/cm³	4	F32		
0x5E	As above, PM4.0		#/cm³	4	F32		
0x5F	As above, PM10.0		#/cm³	4	F32		
0x60	Particulate matter typical particle size		µm	4	F32		
0x61	Gas reading in ppb New sensor ranges available Spring 2021		ppb	5	F32	Note: Gas type byte then 4 bytes for F32 value	
0x62	Corrosion: Sacrificial Metal Thickness	~ 1000nm	nm	5	F32	Note: Coupon/Metal type then 4 bytes for F32 value	
0x63	Corrosion: minimum thickness		nm	3	U16	Note: Coupon/Metal type then 4 bytes for U16 value	
0x64	Corrosion: original thickness		nm	3	U16	Note: Coupon/Metal type then 4 bytes for U16 value	
0x65	Corrosion: percentage of thickness between original thickness (100%) and minimum (0%)		nm	5	F32	Note: Coupon/Metal type then 4 bytes for F32 value	

Table 1: enLink Data Structure

Sensor Data Additional Information

Most sensor data values are self-explanatory, additional information for decoding more complex sensor data is given in the sections below.

enLink Modbus device packet information – 0x0F, 0x10, 0x11

The enLink Modbus data types for Interval and Cumulative values use 5 bytes to encode the item index and value.

- Modbus Exception – standard Modbus exception codes, e.g. Code 2 – Illegal Data Address. (3 bytes)
- Modbus Interval Value – For Modbus data types which do not accumulate, e.g. Voltage, Current Temperature etc. (5 bytes)
- Modbus Cumulative Value – For Modbus data types which are linked to a value which accumulates, e.g. kWh, Volume etc. (5 bytes)

The first byte indicates which of the 32 available Modbus registers is being accessed (0 to 31), followed by the Modbus Value represented as a Float 32 (IEEE754 format). Interval Value types are used for instantaneous values, such as Voltage, Current, Temperature, Pressure etc. Cumulative Values are used for items such as energy consumption and total volume.

Modbus Example

XX	=	Data Type Identifier
XX	=	Index
XX	=	Data Value

Payload Data	10	04	41	BC	7A	E1

This is an interval data value, from configured item number 5. The value is 23.56.

enLink Gas Reading packet information – 0x61

The data sent is 6 bytes in length. For example:

Gas Example

Payload Data	61	19	41	BC	7A	E1

This gas is type 0x19 or 25; Carbon Monoxide. The value is 23.56 ppb.
Other Gas types are:

- 0x17 = Formaldehyde
- 0x18 = VOC (Volatile Organic Compounds)
- 0x19 = Carbon Monoxide
- 0x1A = Chlorine
- 0x1B = Hydrogen
- 0x1C = Hydrogen Sulphide
- 0x1D = Hydrogen Chloride
- 0x1E = Hydrogen Cyanide
- 0x1F = Hydrogen Fluoride
- 0x20 = Ammonia
- 0x21 = Nitrogen Dioxide
- 0x22 = Oxygen
- 0x23 = Ozone
- 0x24 = Sulphur Dioxide

Corrosion Example

Payload Data	62	01	44	58	D0	27

The second byte indicates the Coupon and Metal of the sensor. The example shows Coupon #1 is Copper and the thickness is 867.252 nanometres (equivalent to 8672.52 Ångströms (Å)).
Other Metal types are:

Coupon #1

- 0x00 = Unknown Metal / Error
- 0x01 = Copper
- 0x02 = Silver
- 0x03 = Chromium

Coupon #2

- 0x80 = Unknown Metal / Error
- 0x81 = Copper
- 0x82 = Silver
- 0x83 = Chromium

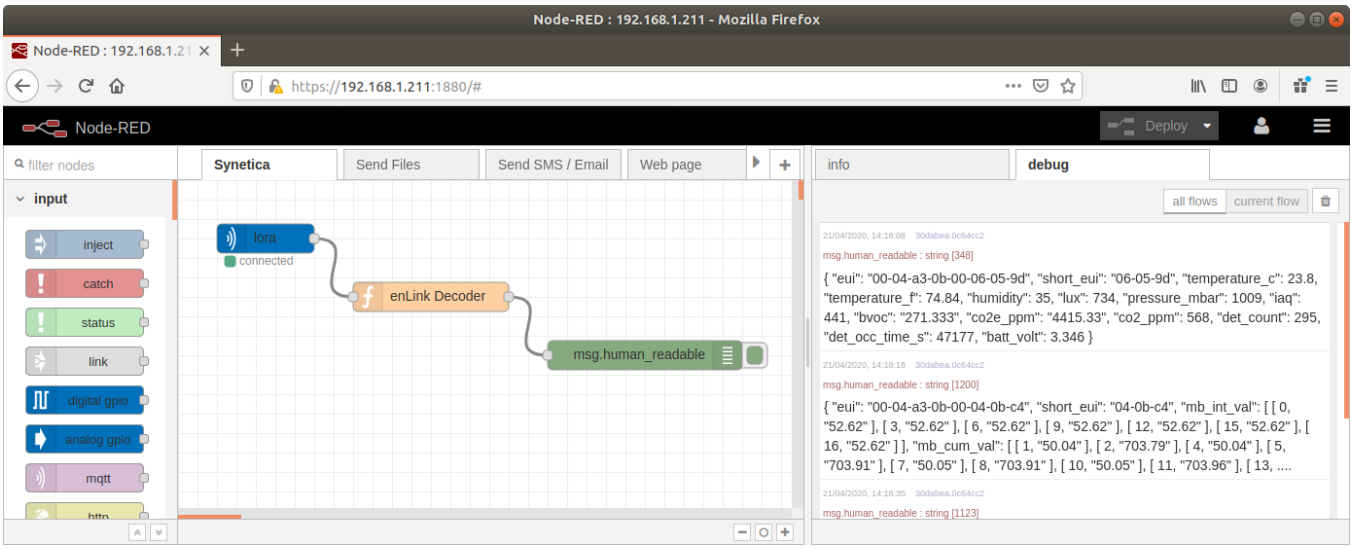
enLink KPI Payload data

Each KPI can use 1 or more bytes to send the value according to the following table.

Data Type	Sensor / Data	Comments	Units	Bytes	Data Type	Data Range
0x40	CPU Temperature	Depreciated April 2020	°C	2	U16	0.00 – 255.99 °C (Packed byte)
0x41	Battery Status	Encoded battery status		1	U8	0=Charging; 1 - 254 (1.8 - 3.3V); 255=Ext Power
0x42	Battery Voltage	Battery voltage level	mV	2	U16	0 -> 3600 mV (3600=Ext Power)
0x43	Rx RSSI	Receive RSSI level	RSSI	2	S16	+- 32767 RSSI
0x44	Rx SNR	Receive SNR level	SNR	1	S8	+- 128 Signal-Noise Ratio
0x45	Rx Count	Downlink message count	count	2	U16	0 -> 65,535
0x46	Tx Time	Time to send message	ms	2	U16	0 -> 65,535
0x47	Tx Power	Transmit power	dBm	1	S8	+- 128
0x48	Tx Count	Uplink message count	count	2	U16	0 -> 65,535
0x49	Power up count	Number of time unit powered up	count	2	U16	0 -> 65,535
0x4A	USB insertions count	Number of times USB activated	count	2	U16	0 -> 65,535
0x4B	Login OK count	Successful logon count	count	2	U16	0 -> 65,535
0x4C	Login fail count	Failed logon count	count	2	U16	0 -> 65,535
0x4D	Fan runtime	Number of seconds that air intake fan has run (AIR only)	s	4	U32	0 -> 136 years
0x4E	CPU Temperature	New for Ver: 4.9	°C	2	S16	-3276.8 --> 3276.7°C

Packet Decoding Examples

JavaScript for decoding enLink Packets is listed below and is available from Synetica. Contact us via support@synetica.net



Node RED Decoding Example.

```
// Used for decoding enLink LoRa Messages
// ------------------------------------------------------------------------------------------
// 22 Mar 2021 (Doc.Ver:4.33 FW Ver:4.33)
//   Add more gas types for latest hardware updates to AIR, ZonePlus (xAQ)
//   Add Corrosion data for thickness of sacrificial metal coupons
// ------------------------------------------------------------------------------------------

if (!msg.eui)
    return null;

// ------------------------------------------------------------------------------------------
// Ignore Port 0 Possible MAC Command
// If there is no payload, there is no need for a port, thus it equals zero
if (msg.port === 0) {
    if (msg.eui) {
        node.warn("Possible MAC Command Received from " + msg.eui);
    } else {
        node.warn("Possible MAC Command Received");
    }
    return null;
}
// Ignore zero payloads
if (msg.payload) {
    if (msg.payload.length === 0) {
        if (msg.eui) {
            node.warn("Zero-length Payload, message ignored from " + msg.eui);
        } else {
            node.warn("Zero-length Payload, message ignored");
        }
        return null;
    }
} else {
    if (msg.eui) {
        node.warn("No Payload, message ignored from " + msg.eui);
    } else {
        node.warn("No Payload, message ignored");
    }
    return null;
}
// ------------------------------------------------------------------------------------------
const ENLINK_TEMP = 0x01;                                   // S16  -3276.8°C -> 3276.7°C (-10..80) [Divide word by 10]
const ENLINK_RH = 0x02;                                     // U8   0 -> 255 %RH (Actually 0..100%)
const ENLINK_LUX = 0x03;                                    // U16  0 -> 65535 Lux
const ENLINK_PRESSURE = 0x04;                               // U16  0 -> 65535 mbar or hPa
const ENLINK_VOC_IAQ = 0x05;                                // U16  0 -> 500 IAQ Index
const ENLINK_O2PERC = 0x06;                                 // U8   0 -> 25.5% [Divide byte by 10]
const ENLINK_CO = 0x07;                                     // U16  0 -> 655.35 ppm (0..100 ppm) [Divide by 100]
const ENLINK_CO2 = 0x08;                                    // U16  0 -> 65535 ppm (0..2000 ppm)
const ENLINK_OZONE = 0x09;                                  // U16  0 -> 6.5535 ppm or 6553.5 ppb (0..1 ppm) [Divide by 10000]
const ENLINK_POLLUTANTS = 0x0A;                             // U16  0 -> 6553.5 kOhm (Typically 100..1500 kOhm) [Divide by 10]
const ENLINK_PM25 = 0x0B;                                   // U16  0 -> 65535 ug/m3 (0..1000 ug/m3)
const ENLINK_PM10 = 0x0C;                                   // U16  0 -> 65535 ug/m3 (0..1000 ug/m3)
const ENLINK_H2S = 0x0D;                                    // U16  0 -> 655.35 ppm (0..100 ppm) [Divide by 100]
const ENLINK_COUNTER = 0x0E;                                // U32  0 -> 2^32
const ENLINK_MB_EXCEPTION = 0x0F;                           // Type Byte + MBID + Exception Code so it's Type + 2 bytes
const ENLINK_MB_INTERVAL = 0x10;                            // Type Byte + MBID + F32 Value - so 6 bytes
const ENLINK_MB_CUMULATIVE = 0x11;                          // Type Byte + MBID + F32 Value - so 6 bytes
const ENLINK_BVOC = 0x12;                                   // F32  ppm Breath VOC Estimate equivalent
const ENLINK_DETECTION_COUNT = 0x13;                        // U32  Counter. Num of detections for PIR/RangeFinder
const ENLINK_OCC_TIME = 0x14;                               // U32  Total Occupied Time (seconds)
const ENLINK_OCC_STATUS = 0x15;                             // U8   Occupied Status. 1=Occupied, 0=Unoccupied
const ENLINK_LIQUID_LEVEL_STATUS = 0x16;                    // U8   Level Status. 1=Detected, 0=Not Detected
const ENLINK_TEMP_PROBE1 = 0x17;                            // S16  As 0x01
const ENLINK_TEMP_PROBE2 = 0x18;                            // S16  As 0x01
const ENLINK_TEMP_PROBE3 = 0x19;                            // S16  As 0x01
const ENLINK_TEMP_PROBE_IN_BAND_DURATION_S_1 = 0x1A;        // U32  Seconds. Time temperature probe 1 has spent in 'in band' zone
const ENLINK_TEMP_PROBE_IN_BAND_DURATION_S_2 = 0x1B;        // U32  Seconds. Time temperature probe 2 has spent in 'in band' zone
const ENLINK_TEMP_PROBE_IN_BAND_DURATION_S_3 = 0x1C;        // U32  Seconds. Time temperature probe 3 has spent in 'in band' zone
const ENLINK_TEMP_PROBE_IN_BAND_ALARM_COUNT_1 = 0x1D;       // U16  Count. Num times in band alarm has activated for probe 1
const ENLINK_TEMP_PROBE_IN_BAND_ALARM_COUNT_2 = 0x1E;       // U16  Count. Num times in band alarm has activated for probe 2
const ENLINK_TEMP_PROBE_IN_BAND_ALARM_COUNT_3 = 0x1F;       // U16  Count. Num times in band alarm has activated for probe 3
const ENLINK_TEMP_PROBE_LOW_DURATION_S_1 = 0x20;            // U32  Seconds. Time probe 1 has spent below low threshold
const ENLINK_TEMP_PROBE_LOW_DURATION_S_2 = 0x21;            // U32  Seconds. Time probe 2 has spent below low threshold
const ENLINK_TEMP_PROBE_LOW_DURATION_S_3 = 0x22;            // U32  Seconds. Time probe 3 has spent below low threshold
const ENLINK_TEMP_PROBE_LOW_ALARM_COUNT_1 = 0x23;           // U16  Count. Num times low threshold alarm has activated for probe 1
const ENLINK_TEMP_PROBE_LOW_ALARM_COUNT_2 = 0x24;           // U16  Count. Num times low threshold alarm has activated for probe 2
const ENLINK_TEMP_PROBE_LOW_ALARM_COUNT_3 = 0x25;           // U16  Count. Num times low threshold alarm has activated for probe 3
const ENLINK_TEMP_PROBE_HIGH_DURATION_S_1 = 0x26;           // U32  Seconds. Time probe 1 has spent above high threshold
const ENLINK_TEMP_PROBE_HIGH_DURATION_S_2 = 0x27;           // U32  Seconds. Time probe 2 has spent above high threshold
const ENLINK_TEMP_PROBE_HIGH_DURATION_S_3 = 0x28;           // U32  Seconds. Time probe 3 has spent above high threshold
const ENLINK_TEMP_PROBE_HIGH_ALARM_COUNT_1 = 0x29;          // U16  Count. Num times high threshold alarm has activated for probe 1
const ENLINK_TEMP_PROBE_HIGH_ALARM_COUNT_2 = 0x2A;          // U16  Count. Num times high threshold alarm has activated for probe 2
const ENLINK_TEMP_PROBE_HIGH_ALARM_COUNT_3 = 0x2B;          // U16  Count. Num times high threshold alarm has activated for probe 3
const ENLINK_DIFF_PRESSURE = 0X2C;                          // F32  +- 5000 Pa
const ENLINK_AIR_FLOW = 0x2D;                               // F32  0 -> 100 m/s
const ENLINK_VOLTAGE = 0x2E;                                // U16  0 -> 65.535V [Divide by 1000]
const ENLINK_CURRENT = 0x2F;                                // U16  0 -> 65.535mA [Divide by 1000]
const ENLINK_RESISTANCE = 0x30;                             // U16  0 -> 65.535kOhm [Divide by 1000]
```

```
const ENLINK_LEAK_DETECT_EVT = 0x31;                    // U8   1 or 0, Leak status on resistance rope
const ENLINK_VIBRATION_EVT = 0x32;                      // U8   1 or 0, vibration event detected

const ENLINK_PRESSURE_TX = 0x3A;                        // U16  Pressure/Depth Transducer (0..50,000 mbar/mm)
const ENLINK_TEMPERATURE_TX = 0x3B;                     // S16  Transducer Temperature -3276.8°C -> 3276.7°C (-10..80)

const ENLINK_CO2E = 0x3F;                               // F32  ppm CO2e Estimate Equivalent

const ENLINK_SOUND_MIN = 0x50;                          // F32  dB(A)
const ENLINK_SOUND_AVG = 0x51;                          // F32  dB(A)
const ENLINK_SOUND_MAX = 0x52;                          // F32  dB(A)
const ENLINK_NO = 0x53;                                 // U16  0 -> 655.35 ppm (0..100 ppm) [Divide by 100]
const ENLINK_NO2 = 0x54;                                // U16  0 -> 6.5535 ppm (0..5 ppm) [Divide by 10000]
const ENLINK_NO2_20 = 0x55;                             // U16  0 -> 65.535 ppm (0..20 ppm) [Divide by 1000]
const ENLINK_SO2 = 0x56;                                // U16  0 -> 65.535 ppm (0..20 ppm) [Divide by 1000]

// Particulate Matter (Advanced Data)
const ENLINK_MC_PM1_0 = 0x57;                           // F32  µg/m³ Mass Concentration
const ENLINK_MC_PM2_5 = 0x58;                           // F32  µg/m³
const ENLINK_MC_PM4_0 = 0x59;                           // F32  µg/m³
const ENLINK_MC_PM10_0 = 0x5A;                          // F32  µg/m³
const ENLINK_NC_PM0_5 = 0x5B;                           // F32  #/cm³ Number Concentration
const ENLINK_NC_PM1_0 = 0x5C;                           // F32  #/cm³
const ENLINK_NC_PM2_5 = 0x5D;                           // F32  #/cm³
const ENLINK_NC_PM4_0 = 0x5E;                           // F32  #/cm³
const ENLINK_NC_PM10_0 = 0x5F;                          // F32  #/cm³
const ENLINK_PM_TPS = 0x60;                             // F32  µm    Typical Particle Size

const ENLINK_GAS_PPB = 0x61;                            // Gas-Type byte + F32 ppb

const ENLINK_CRN_THK = 0x62;                            // Coupon No. + Metal Type byte + F32 nm. Thickness
const ENLINK_CRN_MIN_THK = 0x63;                        // Coupon No. + Metal Type byte + U16 nm. Min Thickness (when depleted)
const ENLINK_CRN_MAX_THK = 0x64;                        // Coupon No. + Metal Type byte + U16 nm. Max/Original Thickness
const ENLINK_CRN_PERC = 0x65;                           // Coupon No. + Metal Type byte + F32 nm.
                                                        //    PERC: Percentage of corrosion between Max(0%) to Min(100%)

// Optional KPI values that can be included in the message
const ENLINK_CPU_TEMP_DEP = 0x40;                       // [DEPRECIATED April 2020. Now 0x4E] 2 bytes 0.0°C -> 255.99°C
const ENLINK_BATT_STATUS = 0x41;                        // U8   0=Charging; 1~254 (1.8 - 3.3V); 255=External Power
const ENLINK_BATT_VOLT = 0x42;                          // U16  0 -> 3600mV (3600mV=External Power)
const ENLINK_RX_RSSI = 0x43;                            // S16  +-32767 RSSI
const ENLINK_RX_SNR = 0x44;                             // S8   +-128 Signal to Noise Ratio
const ENLINK_RX_COUNT = 0x45;                           // U16  0 -> 65535 downlink message count
const ENLINK_TX_TIME = 0x46;                            // U16  0 -> 65535 ms
const ENLINK_TX_POWER = 0x47;                           // S8   +-128 dBm
const ENLINK_TX_COUNT = 0x48;                           // U16  0 -> 65535 uplink message count
const ENLINK_POWER_UP_COUNT = 0x49;                     // U16  0 -> 65535 counts
const ENLINK_USB_IN_COUNT = 0x4A;                       // U16  0 -> 65535 counts
const ENLINK_LOGIN_OK_COUNT = 0x4B;                     // U16  0 -> 65535 counts
const ENLINK_LOGIN_FAIL_COUNT = 0x4C;                   // U16  0 -> 65535 counts
const ENLINK_FAN_RUN_TIME = 0x4D;                       // U32  0 -> 2^32 seconds = 136 years
const ENLINK_CPU_TEMP = 0x4E;                           // S16  -3276.8°C -> 3276.7°C (-10..80) [Divide by 10]

// ------------------------------------------------------------------------------------
// Convert binary value bit to Signed 16 bit
function S16(bin) {
    var num = bin & 0xFFFF;
    if (0x8000 & num)
        num = -(0x010000 - num);
    return num;
}
// Convert binary value bit to Signed 8 bit
function S8(bin) {
    var num = bin & 0xFF;
    if (0x80 & num)
        num = -(0x0100 - num);
    return num;
}
// Useful conversion functions
// S16 -> U16
function U16(ival) {
    if (isNaN(ival) === false) {
        if (ival < 0) {
            ival = ival + 65536;
        }
    }
    return ival;
}
// S32 -> U32 convertIntToDWord
function U32(ival) {
    if (isNaN(ival) === false) {
        if (ival < 0) {
            ival = ival + 4294967296;
        }
    }
    return ival;
}
// U32 -> S32 convertDWordToInt
```

```
function S32(ival) {
    if (isNaN(ival) === false) {
        if (ival > 2147483647) {
            ival = ival - 4294967296;
        }
    }
    return ival;
}
// Utility function
function bytesToHex(bytes) {
    var result = "";
    for (var i = 0; i < bytes.length; i += 1) {
        result += ('0' + (bytes[i]).toString(16).toUpperCase() + ' ').substr(-3);
    }
    return result.trim();
}
// Convert 4 IEEE754 bytes
function fromF32(byte0, byte1, byte2, byte3) {
    var bits = (byte0 << 24) | (byte1 << 16) | (byte2 << 8) | (byte3);
    var sign = ((bits >>> 31) === 0) ? 1.0 : -1.0;
    var e = ((bits >>> 23) & 0xff);
    var m = (e === 0) ? (bits & 0x7fffff) << 1 : (bits & 0x7fffff) | 0x800000;
    var f = sign * m * Math.pow(2, e - 150);
    return f;
}
// Return gas name from gas type byte
function GetGasName(gas_type) {
    switch (gas_type) {
        case 0x17:
            return "Formaldehyde";       //HCHO/CH2O
        case 0x18:
            return "Volatile Organic Compounds";
        case 0x19:
            return "Carbon Monoxide";    //CO
        case 0x1A:
            return "Chlorine";    //Cl2
        case 0x1B:
            return "Hydrogen";    //H2
        case 0x1C:
            return "Hydrogen Sulphide";    //H2S
        case 0x1D:
            return "Hydrogen Chloride";    //HCl
        case 0x1E:
            return "Hydrogen Cyanide";     //HCN
        case 0x1F:
            return "Hydrogen Fluoride";    //HF
        case 0x20:
            return "Ammonia";              //NH2
        case 0x21:
            return "Nitrogen Dioxide";     //NO2
        case 0x22:
            return "Oxygen";     //O2
        case 0x23:
            return "Ozone";      //O3
        case 0x24:
            return "Sulphur Dioxide";      // Sulfur Dioxide (IUPAC) SO2
    }
    return "Unknown";
}
// Corrosion: Return metal name from id byte
function GetCrnMetal(id_byte) {
    var id = (id_byte & 0x7F);
    switch (id) {
        case 0x00:
            return "Unknown";
        case 0x01:
            return "Copper";
        case 0x02:
            return "Silver";
        case 0x03:
            return "Chromium";
    }
    return "Error";
}
// ----------------------------------------------------------------------------------------
// Function to decode enLink Messages
function DecodePayload(data) {
    var cpn;
    var metal;
    var obj = {};
    //obj.eui = msg.eui;
    obj.short_eui = msg.eui.slice(-8);
    var msg_ok = false;
    for (i = 0; i < data.length; i++) {
        switch (data[i]) {
        // Parse Sensor Message Parts
        case ENLINK_TEMP: // Temperature
            obj.temperature_c = (S16((data[i + 1] << 8) | (data[i + 2]))) / 10;
```

```
        obj.temperature_f = ((obj.temperature_c * 9/5) + 32).toFixed(2);
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_RH: // Humidity %rH
        obj.humidity = (data[i + 1]);
        i += 1;
        msg_ok = true;
        break;
    case ENLINK_LUX: // Light Level lux
        obj.lux = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_PRESSURE: // Barometric Pressure
        obj.pressure_mbar = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_VOC_IAQ: // Indoor Air Quality (0-500)
        obj.iaq = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_O2PERC: // O2 percentage
        obj.o2perc = (data[i + 1]) / 10;
        i += 1;
        msg_ok = true;
        break;
    case ENLINK_CO: // Carbon Monoxide
        obj.co_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 100;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_CO2: // Carbon Dioxide
        obj.co2_ppm = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_OZONE: // Ozone ppm and ppb
        obj.ozone_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 10000;
        obj.ozone_ppb = U16((data[i + 1] << 8) | (data[i + 2])) / 10;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_POLLUTANTS: // Pollutants kOhm
        obj.pollutants_kohm = U16((data[i + 1] << 8) | (data[i + 2])) / 10;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_PM25: // Particulates @2.5
        obj.pm25 = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_PM10: // Particulates @10
        obj.pm10 = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_H2S: // Hydrogen Sulphide
        obj.h2s_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 100;
        i += 2;
        msg_ok = true;
        break;

    case ENLINK_COUNTER:
        if (obj.counter) {
            obj.counter.push(
                [ data[i + 1], U32((data[i + 2] << 24) | (data[i + 3] << 16) | (data[i + 4] << 8) | (data[i + 5])) ]);
        } else {
            obj.counter = [
                [ data[i + 1], U32((data[i + 2] << 24) | (data[i + 3] << 16) | (data[i + 4] << 8) | (data[i + 5])) ]
            ];
        }
        i += 5;
        msg_ok = true;
        break;
    case ENLINK_MB_EXCEPTION: // Modbus Error Code
        if (obj.mb_ex) {
            obj.mb_ex.push([ data[i + 1], data[i + 2] ]);
        } else {
            obj.mb_ex = [ [ data[i + 1], data[i + 2] ] ];
        }
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_MB_INTERVAL: // Modbus Interval Read
```

```
        if (obj.mb_int_val) {
            obj.mb_int_val.push([ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(2) ]);
        } else {
            obj.mb_int_val = [ [ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(2) ] ];
        }
        i += 5;
        msg_ok = true;
        break;
    case ENLINK_MB_CUMULATIVE: // Modbus Cumulative Read
        if (obj.mb_cum_val) {
            obj.mb_cum_val.push([ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(2) ]);
        } else {
            obj.mb_cum_val = [ [ data[i + 1], fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(2) ] ];
        }
        i += 5;
        msg_ok = true;
        break;

    case ENLINK_BVOC:     // Breath VOC Estimate equivalent
        obj.bvoc = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(3);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_DETECTION_COUNT:
        obj.det_count = U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_OCC_TIME: // Occupied time in seconds
        obj.occ_time_s = U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_OCC_STATUS: // 1 byte U8, 1 or 0, occupancy status
        obj.occupied = (data[i + 1]) ? true : false;
        i += 1;
        msg_ok = true;
        break;
    case ENLINK_LIQUID_LEVEL_STATUS: // 1 byte U8, 1 or 0, liquid level status
        obj.liquid_detected = (data[i + 1]) ? true : false;
        i += 1;
        msg_ok = true;
        break;

    case ENLINK_TEMP_PROBE1:
        obj.temp_probe_1 = S16((data[i + 1] << 8 | data[i + 2])) / 10;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE2:
        obj.temp_probe_2 = S16((data[i + 1] << 8 | data[i + 2])) / 10;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE3:
        obj.temp_probe_3 = S16((data[i + 1] << 8 | data[i + 2])) / 10;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE_IN_BAND_DURATION_S_1:
        /* Cumulative detection time u32 */
        obj.temp_probe_in_band_duration_s_1 =
            U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE_IN_BAND_DURATION_S_2:
        /* Cumulative detection time u32 */
        obj.temp_probe_in_band_duration_s_2 =
            U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE_IN_BAND_DURATION_S_3:
        /* Cumulative detection time u32 */
        obj.temp_probe_in_band_duration_s_3 =
            U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE_IN_BAND_ALARM_COUNT_1:
        /* In band alarm events u16 */
        obj.temp_probe_in_band_alarm_count_1 = U16((data[i + 1] << 8) | (data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_TEMP_PROBE_IN_BAND_ALARM_COUNT_2:
```

```
    /* In band alarm events u16 */
    obj.temp_probe_in_band_alarm_count_2 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_IN_BAND_ALARM_COUNT_3:
    /* In band alarm events u16 */
    obj.temp_probe_in_band_alarm_count_3 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_LOW_DURATION_S_1:
    /* Cumulative detection time u32 */
    obj.temp_probe_low_duration_s_1 =
        U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
    i += 4;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_LOW_DURATION_S_2:
    /* Cumulative detection time u32 */
    obj.temp_probe_low_duration_s_2 =
        U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
    i += 4;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_LOW_DURATION_S_3:
    /* Cumulative detection time u32 */
    obj.temp_probe_low_duration_s_3 =
        U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
    i += 4;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_LOW_ALARM_COUNT_1:
    /* Low alarm events u16 */
    obj.temp_probe_low_alarm_count_1 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_LOW_ALARM_COUNT_2:
    /* Low alarm events u16 */
    obj.temp_probe_low_alarm_count_2 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_LOW_ALARM_COUNT_3:
    /* Low alarm events u16 */
    obj.temp_probe_low_alarm_count_3 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_HIGH_DURATION_S_1:
    /* Cumulative detection time u32 */
    obj.temp_probe_high_duration_s_1 =
        U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
    i += 4;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_HIGH_DURATION_S_2:
    /* Cumulative detection time u32 */
    obj.temp_probe_high_duration_s_2 =
        U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
    i += 4;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_HIGH_DURATION_S_3:
    /* Cumulative detection time u32 */
    obj.temp_probe_high_duration_s_3 =
        U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
    i += 4;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_HIGH_ALARM_COUNT_1:
    /* High alarm events u16 */
    obj.temp_probe_high_alarm_count_1 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_HIGH_ALARM_COUNT_2:
    /* High alarm events u16 */
    obj.temp_probe_high_alarm_count_2 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
    break;
case ENLINK_TEMP_PROBE_HIGH_ALARM_COUNT_3:
    /* High alarm events u16 */
    obj.temp_probe_high_alarm_count_3 = U16((data[i + 1] << 8) | (data[i + 2]));
    i += 2;
    msg_ok = true;
```

```
        break;

    case ENLINK_DIFF_PRESSURE: // 4 bytes F32, +/- 5000 Pa
        obj.dp_pa = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(3);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_AIR_FLOW: // 4 bytes F32, 0 -> 100m/s
        obj.af_mps = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(3);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_VOLTAGE: // 2 bytes U16, 0 to 10.000 V
        obj.adc_v = U16((data[i + 1] << 8) | (data[i + 2])) / 1000;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_CURRENT: // 2 bytes U16, 0 to 20.000 mA
        obj.adc_ma = U16((data[i + 1] << 8) | (data[i + 2])) / 1000;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_RESISTANCE: // 2 bytes U16, 0 to 10.000 kOhm
        obj.adc_kohm = U16((data[i + 1] << 8) | (data[i + 2])) / 1000;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_LEAK_DETECT_EVT: // 1 byte U8, Leak status changed
        obj.leak_detect_event = (data[i + 1]) ? true : false;
        i += 1;
        msg_ok = true;
        break;
    case ENLINK_VIBRATION_EVT: // 1 byte U8, 1 or 0, vibration event detected
        obj.vibration_event = (data[i + 1]) ? true : false;
        i += 1;
        msg_ok = true;
        break;
    // Pressure Transducer
    case ENLINK_PRESSURE_TX:
        // u16
        obj.pressure_tx_mbar = U16((data[i + 1] << 8 | data[i + 2]));
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_TEMPERATURE_TX:
        //s16 in deci-celcius
        obj.temperature_tx_degc = (S16((data[i + 1] << 8) | (data[i + 2]))) / 10;
        i += 2;
        msg_ok = true;
        break;

    case ENLINK_CO2E: // CO2e Estimate Equivalent
        obj.co2e_ppm = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_SOUND_MIN:
        obj.sound_min_dba = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_SOUND_AVG:
        obj.sound_avg_dba = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_SOUND_MAX:
        obj.sound_max_dba = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_NO: // Nitric Oxide
        obj.no_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 100;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_NO2: // Nitrogen Dioxide scaled at 0-5ppm
        obj.no2_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 10000;
        i += 2;
        msg_ok = true;
        break;
    case ENLINK_NO2_20: // Nitrogen Dioxide scaled at 0-20ppm
        obj.no2_20_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 1000;
        i += 2;
```

```
        msg_ok = true;
        break;
    case ENLINK_SO2: // Sulphur Dioxide 0-20ppm
        obj.so2_ppm = U16((data[i + 1] << 8) | (data[i + 2])) / 1000;
        i += 2;
        msg_ok = true;
        break;

    case ENLINK_MC_PM1_0:
        obj.mc_pm1_0 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_MC_PM2_5:
        obj.mc_pm2_5 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_MC_PM4_0:
        obj.mc_pm4_0 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_MC_PM10_0:
        obj.mc_pm10_0 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_NC_PM0_5:
        obj.nc_pm0_5 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_NC_PM1_0:
        obj.nc_pm1_0 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_NC_PM2_5:
        obj.nc_pm2_5 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_NC_PM4_0:
        obj.nc_pm4_0 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;
    case ENLINK_NC_PM10_0:
        obj.nc_pm10_0 = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_PM_TPS:
        obj.pm_tps = fromF32(data[i + 1], data[i + 2], data[i + 3], data[i + 4]).toFixed(2);
        i += 4;
        msg_ok = true;
        break;

    case ENLINK_GAS_PPB:
        obj.gas_ppb_type = data[i + 1];
        obj.gas_ppb_name = GetGasName(data[i + 1]);
        obj.gas_ppb_val = fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(2);
        // As Array
        if (obj.gas_ppb) {
            obj.gas_ppb.push([ data[i + 1], obj.gas_ppb_val ]);
        } else {
            obj.gas_ppb = [ [ data[i + 1], obj.gas_ppb_val ] ];
        }
        i += 5;
        msg_ok = true;
        break;

    case ENLINK_CRN_THK:
        // Coupon is either 1 or 2. Bit 7 set for Coupon 2
        cpn = (data[i + 1] & 0x80) === 0 ? 1 : 2;
        metal = GetCrnMetal(data[i + 1]);
        // Thickness in nanometres
        var thk_nm = fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(2);
        // As Array
        if (obj.crn_thk_nm) {
            obj.crn_thk_nm.push([ cpn, metal, thk_nm ]);
        } else {
            obj.crn_thk_nm = [ [ cpn, metal, thk_nm ] ];
        }
        i += 5;
```

```
            msg_ok = true;
            break;

        case ENLINK_CRN_MIN_THK:
            cpn = (data[i + 1] & 0x80) === 0 ? 1 : 2;
            metal = GetCrnMetal(data[i + 1]);
            // Minimum thickness of metal
            var min_nm = U16((data[i + 2] << 8) | (data[i + 3]));
            // As Array
            if (obj.crn_min_nm) {
                obj.crn_min_nm.push([ cpn, metal, min_nm ]);
            } else {
                obj.crn_min_nm = [ [ cpn, metal, min_nm ] ];
            }
            i += 3;
            msg_ok = true;
            break;

        case ENLINK_CRN_MAX_THK:
            cpn = (data[i + 1] & 0x80) === 0 ? 1 : 2;
            metal = GetCrnMetal(data[i + 1]);
            // Original thickness of metal
            var max_nm = U16((data[i + 2] << 8) | (data[i + 3]));
            // As Array
            if (obj.crn_max_nm) {
                obj.crn_max_nm.push([ cpn, metal, max_nm ]);
            } else {
                obj.crn_max_nm = [ [ cpn, metal, max_nm ] ];
            }
            i += 3;
            msg_ok = true;
            break;

        case ENLINK_CRN_PERC:
            cpn = (data[i + 1] & 0x80) === 0 ? 1 : 2;
            metal = GetCrnMetal(data[i + 1]);
            // Corrosion of coupon in percentage from Max(0%) to Min(100%)
            var perc = fromF32(data[i + 2], data[i + 3], data[i + 4], data[i + 5]).toFixed(3);
            // As Array
            if (obj.crn_perc) {
                obj.crn_perc.push([ cpn, metal, perc ]);
            } else {
                obj.crn_perc = [ [ cpn, metal, perc ] ];
            }
            i += 5;
            msg_ok = true;
            break;

        // < ---------------------------------------------------------------------------->
        // Optional KPIs
        case ENLINK_CPU_TEMP_DEP:    // Optional from April 2020
            obj.cpu_temp_dep = data[i + 1] + (Math.round(data[i + 2] * 100 / 256) / 100);
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_CPU_TEMP:    // New for April 2020 Ver: 4.9
            obj.cpu_temp = (S16((data[i + 1] << 8) | (data[i + 2]))) / 10;
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_BATT_STATUS:
            obj.batt_status = data[i + 1];
            i += 1;
            msg_ok = true;
            break;
        case ENLINK_BATT_VOLT:
            obj.batt_volt = U16((data[i + 1] << 8) | (data[i + 2])) / 1000;
            obj.batt_mv = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_RX_RSSI:
            obj.rx_rssi = S16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_RX_SNR:
            obj.rx_snr = S8(data[i + 1]);
            i += 1;
            msg_ok = true;
            break;
        case ENLINK_RX_COUNT:
            obj.rx_count = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_TX_TIME:
            obj.tx_time_ms = U16((data[i + 1] << 8) | (data[i + 2]));
```

```
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_TX_POWER:
            obj.tx_power_dbm = S8(data[i + 1]);
            i += 1;
            msg_ok = true;
            break;
        case ENLINK_TX_COUNT:
            obj.tx_count = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_POWER_UP_COUNT:
            obj.power_up_count = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_USB_IN_COUNT:
            obj.usb_in_count = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_LOGIN_OK_COUNT:
            obj.login_ok_count = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_LOGIN_FAIL_COUNT:
            obj.login_fail_count = U16((data[i + 1] << 8) | (data[i + 2]));
            i += 2;
            msg_ok = true;
            break;
        case ENLINK_FAN_RUN_TIME:
            obj.fan_run_time_s =
                U32((data[i + 1] << 24) | (data[i + 2] << 16) | (data[i + 3] << 8) | (data[i + 4]));
            i += 4;
            msg_ok = true;
            break;

        default: // something is wrong with data
            i = data.length;
            msg_ok = true;
            break;
        }
    }
    if (msg_ok) {
        return obj;
    } else {
        return null;
    }
}
// ----------------------------------------------------------------------------------------
var res = DecodePayload(msg.payload);
if (res !== null) {
    msg.hex = bytesToHex(msg.payload);
    msg.payload = res;  // use for further function processing
    msg.human_readable = JSON.stringify(res, null, 4);
    return msg;
}
return null;
```

JavaScript to decode enLink messages for Node-RED and display values in the debug pane

Document Revision History

Date	Revision	Changes
1 Oct 2017	1.0	• Initial Document
22 Jan 2018	1.1	• Corrected Data-Range example for Ozone sensor to clarify units • Added Hydrogen Sulphide Sensor (H_2S) • Changed units for Pollutant sensor from ppm to resistance kΩ to allow user to define own results • Updated JavaScript sample to reflect changes • AQM firmware 1.3
2 Feb 2018	1.2	• Add LUX sensor configuration parameters for offset and scale • AQM firmware 1.4
28 Nov 2018	1.3	• Changed JavaScript example to handle the optional KPI messages • All firmware from 1.14
25 Feb 2019	2.0	• Improve explanation on data value conversion • All firmware from 2.0
07 May 2019	2.1	• Added new data types and KPI packet information
08 July 2019	2.2	• Added temperature probe alarm values and updated JavaScript
28 Aug 2019	2.3	• Updated temperature probe format from F32 to S16
01 Feb 2020	2.4	• Updated to latest JavaScript. Added NO, NO_2 (0-5ppm), Sound. • Changed name for 0x13/0x14
21 April 2020	2.5	• Amended CPU Temperature, and used 'normal' method for code 0x4E • Added 0x15 for Occupancy Status • Fix Event decoder logic for 0x31 and 0x32 • Renamed TYPE to ENLINK for future development with DataStream
05 May 2020	2.6	• Added 0x55 and 0x56 for NO_2 (0-20ppm) and SO_2
08 May 2020	2.7	• Changed data types for 0x2C and 0x2D to F32 – Diff Pressure and Air Flow
23 Jun 2020	4.15	• Change document version to match enLink Firmware Version • Add 10 types for advanced new indoor/outdoor Air Quality units with Particulate Matter sensor 'enl-aq-p+'
22 Oct 2020	4.21	• Add type 0x16 for liquid level detection – used with Pura Sanitisers Hand Sanitiser station. • Fix the normally signed variables to unsigned where needed.
10 Mar 2021	4.31	• Add type 0x61 for multiple gasses for latest hardware updates to AIR and ZonePlus
24 Mar 2021	4.33	• Add 4 message types (0x62 – 0x65) for Corrosion: thickness of sacrificial metal coupons • Add more gas types for message type 0x61