iMETOS LoRa
Data payload structure

Pessl Instruments, GmbH

Version 1.0, 06-2018

Content

1. Scope of this document

This document describes the data payload structure used on the iMETOS LoRa weather station. Sample Python scripts for parsing the datagram are available on the iMETOS LoRa website.

2. Parsing the data from the payload versus API data access

The preferred way which we support is retrieving the data via our API system from the Fieldclimate platform.

When the user wants to parse the sensor data from the raw data payload delivered from the iMETOS LoRa weather station, he will also need the API access in order to get the description of sensors. If the user has an access to our API sensor description, he can also retrieve the sensors data and other calculated values from the API and there is no need for further custom parsing of the raw data.

The data payload structure can change in the upcoming firmware versions without prior notification.

Example: it can happen, that the user has an iMETOS LoRa weather station with firmware version 1.05 which supports the data payload structure version 1.0 revision D and in the future the client can get the iMETOS LoRa weather station with firmware version 1.1, where the data payload structure version 2.0 will be integrated.

3. iMETOS LoRa firmware versions with data payload structure

When parsing the raw data, the user needs to know, which data payload structure is used in regard of the firmware used on the device. This table explains the connection between them:

iMETOS LoRa firmware version	Data payload structure version	Sample Python script
iMETOSLoRaV100_v105.X.production.hex	Datagram structure v100 rev D	iMETOS_LP_v100_revC.py
iMETOSLoRaV100_v104.X.production.hex	Datagram structure v100 rev D	iMETOS_LP_v100_revC.py
iMETOSLoRaV100_v103.X.production.hex	Datagram structure v100 rev C	iMETOS_LP_v100_revC.py

4. Data payload structure

4.1 Datagram structure v100 rev C

Note: if the user does not set the correct data time on the station, it will send the data starting with the year 2000. In this case, we suggest you use the UTS time of received data in your parser.

```
Main structure of PI IoT datagram strcuture:
|            <HEADER>            |           <SENSOR DATA>           |
 <CRC> <msgID> <info> <timestamp> <data packet 1> ... <data packet X>

Packets description:

<CRC>: 2 Bytes
offset  0 / 2 bytes: CR = 16-bit CRC - Checksum of all following Bytes.

<msgID>: 2 Bytes
offset  0 / 1 byte:  MN = Incrementing message number
offset  1 / 1 byte:  SI = unique ID of the datagram structure (0x01 = datagram structure
version 1.xx)
```

<info>: 11 Bytes
offset 0 / 1 byte: DI = unique PI device ID
offset 1 / 2 bytes: HW = HW version (LSB = version 0.01)
offset 3 / 2 bytes: FW = FW version (LSB = version 0.01)
offset 5 / 2 bytes: DS = device status - NOT YET USED
offset 7 / 4 bytes: SN = 32-bit device serial number (the same format like at iMetos)

<timestamp>: 6 Bytes
offset 0 / 1 byte: YY = years part of the date, binary coded decimal
offset 1 / 1 byte: MM = months part of the date, binary coded decimal
offset 2 / 1 byte: DD = days part of the date, binary coded decimal
offset 3 / 1 byte: ss = seconds part of the 24-hours time, binary coded decimal
offset 4 / 1 byte: mm = minutes part of the 24-hours time, binary coded decimal
offset 5 / 1 byte: hh = hours part of the 24-hours time, binary coded decimal

<data packet>: the length is variable
offset 0 / 2 bytes: SC = Sensor code. This unique code defines the number of sensor values/attributes and their size. See the document about the list of all sensor codes used for PI IoT datagram structure.
offset 2 / 1 byte: CH = unique channel number
offset 3 / variable number of bytes = sensor values (average (AVG), minimal (MIN), maximal (MAX), summary (SUM), last (LST) or other sensor attributes)

* all multi-byte binary fields are ordered little endian

AN EXAMPLE OF ABOVE DEFINED DATAGRAM

CR = 0x0000
MN = 25
SI = 1
DI = 2
HW = 2.01
FW = 1.05
DS = 0x0001
SN = 0320001A
YY = 0x17
MM = 0x06
DD = 0x30
ss = 0x00
mm = 0x15
hh = 0x12

Sensors	Nm.	sensor code	ch.	val. size	AVG	MIN	MAX	LST	SUM
Battery vol.	SC1	0x0007	1	2 Bytes				6250 mV	
Solar panel	SC2	0x001E	2	2 Bytes				5500 mV	
Solar rad.	SC3	0x0258	6	2 Bytes	1538 W/m2				
Water met. 1L	SC4	0x002B	4	4 Bytes					12601 L
HC Rel. hum.	SC5	0x01FB	8	2 Bytes	45.75	42.11	46.12		
HC Air tem.	SC6	0x01FA	7	2 Bytes	-1.65	-3.01	2.58		

Datagram structure (61 Bytes as example):

CR	MN	SI	DI	HW	FW	DS	SN	YY	MM	DD	ss	mm	hh	SC1	CH	LST	SC2	CH	LST	SC3	CH	AVG	SC4	CH	SUM	SC5	CH	AVG	MIN	MAX	SC6	CH	AVG	MIN	MAX
59 12	19	01	02	C9 00	69 00	01 00	1A 00 20 03	17	06	30	00	15	12	07	00	01	6A 18	1E 00	02	7C 15	58	02	06	02	06 2B 00	04	3A EC	01 00	FB 01	08	DF 11	73 10	04	12	FA 01

07 5A FF D2 FE 02 01

CRC-16 ALGORITHM USING POLYNOM 0x8005:

```
const rom unsigned far int CRC16tbl [ 256 ] = {
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
```

```
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040 };

unsigned short addCRC(unsigned short crc, unsigned char dataByte)
{
    unsigned int crc16int;
    memcpypgm2ram(&crc16int, &CRC16tbl[ ( crc & 0xFF ) ^ dataByte ], 2);
    return ( ( crc >> 8 ) ^ crc16int );
}

unsigned short crc16_calc(char *pBuffer, unsigned short length)
{
    unsigned short crc16 = 0;
    while (length--)
       crc16 = addCRC( crc16, *pBuffer++);

    return crc16;
}
```

4.2 Datagram structure v100 rev D

Note: if the user does not set the correct data time on the station, it will send the data starting with the year 2000. In this case, we suggest you use the UTS time of received data in your parser.

```
Main structure of PI IoT datagram strcuture:
|            <HEADER>            |           <SENSOR DATA>           |
<CRC> <msgID> <info> <timestamp> <data packet 1> ... <data packet X>



Packets description:

<CRC>: 2 Bytes
offset  0 / 2 bytes: CR = 16-bit CRC - Checksum of all following Bytes.

<msgID>: 2 Bytes
offset  0 / 1 byte:  MN = Incrementing message number
offset  1 / 1 byte:  SI = unique ID of the datagram structure (0x01 = datagram structure
version 1.xx)

<info>: 11 Bytes
offset  0 / 1 byte:  DI = unique PI device ID
offset  1 / 2 bytes: HW = HW version (LSB = version 0.01)
offset  3 / 2 bytes: FW = FW version (LSB = version 0.01)
offset  5 / 2 bytes: DS = device status
```

offset 7 / 4 bytes: SN = 32-bit device serial number (the same format like at iMetos)

<timestamp>: 6 Bytes
offset 0 / 1 byte: YY = years part of the date, binary coded decimal
offset 1 / 1 byte: MM = months part of the date, binary coded decimal
offset 2 / 1 byte: DD = days part of the date, binary coded decimal
offset 3 / 1 byte: ss = seconds part of the 24-hours time, binary coded decimal
offset 4 / 1 byte: mm = minutes part of the 24-hours time, binary coded decimal
offset 5 / 1 byte: hh = hours part of the 24-hours time, binary coded decimal

<data packet>: the length is variable

offset 0 / 2 bytes: SC = Sensor code. This unique code defines the number of sensor values/attributes and their size.
                     See the document about the list of all sensor codes used for PI IoT datagram structure.
offset 2 / 1 byte: CH = unique channel number
offset 3 / variable number of bytes = sensor values (average (AVG), minimal (MIN), maximal (MAX), summary (SUM), last (LST) or other sensor attributes)

* all multi-byte binary fields are ordered little endian

****

Device status (DS) definition:

- 16-bit length device status number contains status of IoT device, events, other paramters or requests.

bit0:   1 - Request for all current settings incl. timestamp.
                 0 - No request.
bit1-bit15: 0 (not used, all is zero)

****

AN EXAMPLE OF ABOVE DEFINED DATAGRAM

CR = 0x1259

MN = 25
SI = 1

DI = 2
HW = 2.01
FW = 1.05
DS = 0x0001 (bit0 = 1: Request for response with settings)
SN = 0320001A

YY = 0x17
MM = 0x06
DD = 0x30
ss = 0x00
mm = 0x15
hh = 0x12

Sensors	Nm.	sensor code	channel	value size	AVG	MIN	MAX	LST	SUM
Battery vol.	SC1	0x0007	1	2 Bytes					6250 mV
Solar panel	SC2	0x001E	2	2 Bytes					5500 mV
Solar rad.	SC3	0x0258	6	2 Bytes	1538 W/m2				
Water met. 1	LSC4	0x002B	4	4 Bytes					126010 L
HC Rel. hum.	SC5	0x01FB	8	2 Bytes	45.75	42.11	46.12		
HC Air temp.	SC6	0x01FA	7	2 Bytes	-1.65	-3.01	2.58		

Datagram structure (61 Bytes in HEX as example):

```
 CR  |MN|SI|DI| HW  | FW  | DS  |    SN     |YY|MM|DD|ss|mm|hh| SC1 |CH| LST | SC2 |CH| LST
| SC3 |CH| AVG | SC4 |CH|    SUM    | SC5 |CH| AVG | MIN | MAX | SC6 |CH| AVG | MIN | MAX |
59 12 19 01 02 C9 00 69 00 01 00 1A 00 20 03 17 06 30 00 15 12 07 00 01 6A 18 1E 00 02 7C
15 58 02 06 02 06 2B 00 04 3A EC 01 00 FB 01 08 DF 11 73 10 04 12 FA 01 07 5A FF D2 FE 02
01

*********************************************************************************************


CRC-16 ALGORITHM USING POLYNOM 0x8005:

const rom unsigned far int CRC16tbl [ 256 ] = {
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040 };

unsigned short addCRC(unsigned short crc, unsigned char dataByte)
{
    unsigned int crc16int;
    memcpypgm2ram(&crc16int, &CRC16tbl[ ( crc & 0xFF ) ^ dataByte ], 2);
    return ( ( crc >> 8 ) ^ crc16int );
}

unsigned short crc16_calc(char *pBuffer, unsigned short length)
{
    unsigned short crc16 = 0;
    while (length--)
       crc16 = addCRC( crc16, *pBuffer++);

    return crc16;
}
```

5. Guide revisions

GUIDE VERSION	MODIFICATIONS
1.00	- First release of the document.