タグの構成
タグは
　　クライアントID（＝MAC ADDRESS）/ センサ番号 /データ形式
で構成される。

タグレベル1　クライアントID
　　クライアントIDは必ずMAC ADDRESSでなければならない。
　　ということは、一つのハードウェアには、一つのクライアントしか載せられないという制限を持つ。

タグレベル２　センサ番号（数値ふた桁を想定）
　　一つのハードウェアに複数のセンサが乗る場合、01, 02 といったユニット番号を有する。

タグレベル３　データ形式
　　一つのセンサは基本的に一つのデータ形式（加速度ならば acc01とか）を送ってくるが、時々 info が送られてくることになる。

データ形式一覧

info

概要
センサ情報。ユニット単位ではなく、センサ単位で送られてくる？そうだとすると、厳密にはネットワーク関係の情報は無くても良いことにはなる。

内容
JSON形式で以下のキーを含む

キー	項目
name	センサ名
sensor	データ形式名
time	時刻 (“yyyy-MM-dd HH:mm:ss”)
location	設置場所
interface	ネットワークインターフェイス名
hardwareaddress	MACアドレス
hostaddress	IPアドレス

コードサンプル
```
import org.json.JSONObject;

public void messageArrived(String topic, MqttMessage message) throws MqttException {
        byte[] buffer = message.getPayload();
        JSONObject info = new JSONObject(new String(buffer));
}
```

コードサンプル **(javascript)**
```
function on MessageArrived(message) {
        var str = new TextDecoder(“utf-8”).decode(message.payloadBytes);
        var info = JSON.parse(str);
}
```

acc01

概要
加速度センサADXL355によって計測した加速度データ

内容

	項目	単位	サイズ	型
1	UNIX時刻	ミリ秒	8byte	long
2	データサイズ	個	4byte	int
3	加速度	※1	データサイズ×3byte	※2

※1: 980/256000倍するとgalになる
※2: 2の補数フォーマットの20bitデータ。3byte=24bitのうち、上位20bitを使用。ビッグエンディアン

コードサンプル

```
public void messageArrived(String topic, MqttMessage message) throws MqttException {
	byte[] buffer = message.getPayload();
	ByteBuffer bct = ByteBuffer.wrap(Arrays.copyOf(buffer, 8));
	long ct = bct.getLong(); // UNIX時刻
	ByteBuffer bsize = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 8, 12));
	int size = bsize.getInt(); // データサイズ
	double []acc = new double[size]; // 加速度
	ByteBuffer bacc = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 12, 12+3*size));
	for (int i=0; i < size; i++) {
		int accval = convertAcc(bacc, 3*i);
		acc[i] = accval*980.0/256000.0;
	}
}

public int pow19 = (int) Math.pow(2, 19);
public int pow20 = (int) Math.pow(2, 20);
public int convertAcc(ByteBuffer bb, int index) {
	int a = Byte.toUnsignedInt(bb.get(index)) * 4096;// 最上位バイト
	int b = Byte.toUnsignedInt(bb.get(index + 1)) * 16;// 中間バイト
	int c = Byte.toUnsignedInt(bb.get(index + 2)) / 16;// 最下位バイト
	int ans = a + b + c;
	if (ans > pow19) {
		ans -= pow20;
	}
	return ans;
}
```

acc02：加速度

概要
加速度センサADXL355によって計測した加速度データ

内容

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	この後のデータバイト数	バイト	4byte	int	下記3と4の合計バイト数
3	データ個数	個	4byte	int	必要数、繰返し。
4	加速度	※1	データ個数 ×3byte	※2	

※1: 980/256000倍するとgalになる
※2:　2の補数フォーマットの20bitデータ。3byte=24bitのうち、上位20bitを使用。ビッグエンディアン
2020/9/20追記。下位4ビットの情報は下記の通り。
bit0 = Xフラグ。Xだと1になる。ADXLから渡されるものがそのまま送られてくる。
bit1 = EMPTY フラグ。fifo が空だと1になる。ADXLから渡されるが、これが1のデータはプログラムが送ってこないので、これが1になることは無いはず。
bit2 = 未使用。（Acc02Processorが使用。メッセージの最初のデータだと1にする。）
bit3 = 2020/9/20に修正した app.py では fifo overrunを検知すると、これを1する。

備考
周波数は定義されていない。infoで確認すること。
ある一つの方向の加速度データが「データ1個」である。
通常はデータ3個である一瞬の3軸加速度データとなる。
ただし、データの最初がかならず X とは限らない。（なるべくXにして欲しい。）

コードサンプル

```
public void messageArrived(String topic, MqttMessage message) throws MqttException {
        ByteBuffer bb=ByteBuffer.wrap(Arrays.copyOf(message.getPayload()));
        long unitTimeMillis = bb.getLong(); // [1] unittime
        LocalDateTime unitTime =
                LocalDateTime.ofInstant(Instant.ofEpochMilli(unitTimeMillis), ZoneId.systemDefault());

      int recordLength = bb.getInt(); //  [2] bytelength
      log.info("recordLength=" + recordLength);          while (bb.hasRemaining()) {
        int fifoLength = bb.getInt();   //[r3]length(/3, int 4bytes)
        for (int i = 0; i < fifoLength / 3; i++) {// [r4] data (9bytes)
                int x = convertAcc(bb, true); // X方向加速度(整数値)
                int y = convertAcc(bb, false); // Y方向加速度(整数値)]
                int z = convertAcc(bb, false);// Z方向加速度(整数値)
        }
      }
}

 public static final int pow19 = (int) Math.pow(2, 19);
   public static final int pow20 = (int) Math.pow(2, 20);

   public static int convertAcc(ByteBuffer bb, boolean xaxis) {
      int a = Byte.toUnsignedInt(bb.get()) * 4096;// 最上位バイト
```

```
    int b = Byte.toUnsignedInt(bb.get()) * 16;// 中間バイト
    byte c0 = bb.get();
    switch (c0 & (byte) 0x03) {
        case 0:
            if (xaxis) {
                log.warning("last two bit = 0 for x");
            }
            break;
        case 1:
            if (!xaxis) {
                log.warning("last two bit = 1 for or y");
            }
            break;
        case 2:
        case 3:
            log.warning("empty bit is set");
            break;
        default:
    }

    int c = Byte.toUnsignedInt(c0) / 16;// 最下位バイト
    int ans = a + b + c;
    if (ans > pow19) {
        ans -= pow20;
    }
    return ans;
}
```

acc01/fft

概要
acc01のフーリエ振幅スペクトル

内容

項目	単位	サイズ	型
データサイズ	個	2byte	short
周波数刻み	[Hz]	4byte	float
フーリエ振幅スペクトル※1	[gal * s]	3方向×4byte×データサイズ	float

※1:　NS、EW、UDの3個セットがデータサイズ分並んでいる。
　　(NS0 EW0 UD0 NS1 EW1 UD1 NS2 EW2 UD2 …)
すべてリトルエンディアン

コードサンプル

```
public void messageArrived(String topic, MqttMessage message) throws MqttException {
	byte[] buffer = message.getPayload();
	ByteBuffer bsize = ByteBuffer.wrap(Arrays.copyOf(buffer, 2));
	bsize.order(ByteOrder.LITTLE_ENDIAN);
	short size = bsize.getShort();

	ByteBuffer bdf = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 2, 6));
	bdf.order(ByteOrder.LITTLE_ENDIAN);
	float df = bdf.getFloat();

	float[][] fft = new float[size/2][3];
	ByteBuffer bfft = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 6, buffer.length));
	bfft.order(ByteOrder.LITTLE_ENDIAN);
	for (int i = 0; i < size/2; i++) {
		fft[i][0] = bfft.getFloat(); // NS
		fft[i][1] = bfft.getFloat(); // EW
		fft[i][2] = bfft.getFloat(); // UD
	}
}
```

コードサンプル **(javascript)**

```
function on MessageArrived(message) {
	var data = message.payloadBytes;
	var dv = new DataView(data.buffer, data.byteOffset);
	var size = dv.getInt16(0, true);
	var df = dv.getFloat32(2, true);
	var datans = [];
	var dataew = [];
	var dataud = [];
	for (var i = 0; i < size/2; i++) {
		datans.push({freq: df*i, amp: dv.getFloat32(6+i*12, true)});
		dataew.push({freq: df*i, amp: dv.getFloat32(10+i*12, true)});
```

```
                dataud.push({freq: df*i, amp: dv.getFloat32(14+i*12, true)});
        }
}
```

acc01/jma

概要
acc01の気象庁計測震度

内容

項目	単位	サイズ	型
計測震度		4byte	float

コードサンプル
```
public void messageArrived(String topic, MqttMessage message) throws MqttException {
	byte[] buffer = message.getPayload();
	ByteBuffer bjma = ByteBuffer.wrap(Arrays.copyOf(buffer, 4));
	bjma.order(ByteOrder.LITTLE_ENDIAN);
	float jma = bjma.getFloat();
}
```

コードサンプル **(javascript)**
```
function on MessageArrived(message) {
	var data = message.payloadBytes;
	var dv = new DataView(data.buffer, data.byteOffset);
	var jma = dv.getFloat32(0, true); // <- little endian
}
```

ir01：人感センサデータ

概要
赤外線センサHC-SR501によって計測した人感データ

内容（旧バージョン、緑ヶ丘採用バージョン以外はこれを使っていると思われる。）

項目	単位	サイズ	型
検出有無	--	1byte	byte(*1)

内容（新バージョン、緑ヶ丘採用バージョンはこれに従っている。

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	この後のデータバイト数	バイト	4byte	int	下記3のバイト数。ir01では”1”固定。
3	検出有無	--	1byte	byte	

説明
検出有無

7(MSB)	6	5	4	3	2	1	0(LSB)
未使用	未使用	未使用	未使用	未使用	未使用	IR02	IR01

検出されていれば1、されていなければ0となる。

コードサンプル
重要な部分は赤いところだけ。なおこれは旧バージョン。新バージョンでは [0]のところを[12]にする必要がある。

```
public void messageArrived(String tag, MqttMessage message) throws Exception {
  try {
    log.info(tag);
    byte[] buffer = message.getPayload();
    Statement st = con.createStatement();
    st.executeUpdate("create table if not exists \""
      + tag
      + "\" (_NO identity, TIMESTAMP timestamp, LEFT boolean, RIGHT boolean)");
    st.close();

    PreparedStatement ps = con.prepareStatement(
      "insert into \"" + tag + "\" (TIMESTAMP, LEFT, RIGHT) values (?,?,?)");
    Timestamp timestamp = Timestamp.from(Instant.now());
```

```
            ps.setTimestamp(1, timestamp);
            ps.setBoolean(2, ((buffer[0] & 1) != 0));
            ps.setBoolean(3, ((buffer[0] & 2) != 0));
            ps.execute();
            ps.close();
        } catch (Exception e) {
            Logger.getLogger(SubscriberIr01.class.getName()).log(Level.INFO, "message", e);
            throw e;
        }
    }
```

array01

概要
赤外線アレイセンサAMG88によって計測した温度分布データ

内容

項目	単位	サイズ	型
UNIX時刻	ミリ秒	8byte	long
データサイズ	個	4byte	int
センサ温度	※1	2byte	※3
計測温度	※2	128byte	※3

※1: 0.0625倍すると℃になる
※2: 0.25倍すると℃になる
※3: 2byte×64ワード。2byte=16bitのうち、下位11bitが値、下から12bit目が符号。リトルエンディアン

コードサンプル

```
public void messageArrived(String topic, MqttMessage message) throws MqttException {
        byte[] buffer = message.getPayload();
        ByteBuffer bct = ByteBuffer.wrap(Arrays.copyOf(buffer, 8));
        long ct = bct.getLong(); // UNIX時刻
        ByteBuffer bsize = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 8, 12));
        int size = bsize.getInt(); // データサイズ
        double therm = convert(buffer, 12) * 0.0625; // センサ温度
        double []temp = new double[64]; // 計測温度
        for (int i=0; i < 64; i++) {
                int tempval = convert(buffer, 14+2*i);
                if ((buffer[14+2*i+1] & 0x08) > 0) {
                        temp[i] = tempval*(-0.25);
                } else {
                        temp[i] = tempval * 0.25;
                }
        }
}
```

```
public int convert(byte[] b, int index) {
	return ((0x07 & b[index+1]) << 8) | (0xff & b[index]);
}
```

array02

概要
赤外線アレイセンサAMG88によって計測した温度分布データ

内容

項目	単位	サイズ	型
UNIX時刻	ミリ秒	8byte	long
データサイズ	個	4byte	int
センサ温度	℃	4byte	float
計測温度	℃	256byte	float

コードサンプル

```
public void messageArrived(String topic, MqttMessage message) throws MqttException {
	byte[] buffer = message.getPayload();
	ByteBuffer bct = ByteBuffer.wrap(Arrays.copyOf(buffer, 8));
	long ct = bct.getLong(); // UNIX時刻
	ByteBuffer bsize = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 8, 12));
	int size = bsize.getInt(); // データサイズ
	ByteBuffer bstemp = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 12, 16));
	float stemp = bstemp.getFloat(); // センサ温度
	float []temp = new float[64]; // 計測温度
	for (int i=0; i < 64; i++) {
		ByteBuffer btemp = ByteBuffer.wrap(Arrays.copyOfRange(buffer, 16, 16+4*i));
		float temp = btemp.getFloat();
	}
}
```

str01 : ひずみ

概要
ひずみデータ

内容
すべてbigendian (MSBから送られてくる）

	項目	単位	サイズ	型	
1	送信時UNIX時刻	ミリ秒	8bytes	long	送信直前のユニット時刻
2	この後のデータバイト数	バイト	4bytes	int	下記3と4の合計バイト数
3	データ記録開始UNIX時刻	ミリ秒	8bytes	long	前回の送信時UNIX時刻と同じ
4	ひずみデータ		4bytes （以後繰返し）	int	24ビット整数値

（「データバイト数」−8）÷4　によりひずみデータの個数が計算できる。

「ひずみデータ」の構成
一つのひずみデータは4バイトから構成されるが、下位3バイトがひずみデータを示しており、最上位1ビットはフラグが入ってくる。

MSB			**LSB**
フラグ	←ひずみデータ	←ひずみデータ→	ひずみデータ→

フラグ

GPIO21	GPIO20	reserved	reserved	reserved	reserved	reserved	reserved

mcp01: ひずみゼロ調データ

概要
ひずみデータのゼロ調整データ。
チャンネルごとに、set, get あるいは balance すると送ってくる。

内容
すべてbigendian (MSBから送られてくる）

	項目	単位	サイズ	型	
1	送信時UNIX時刻	ミリ秒	8bytes	long	送信直前のユニット時刻
2	この後のデータバイト数	バイト	4bytes	int	下記3と4の合計バイト数
3	ボードバージョン		4bytes	char	ひずみボードのバージョン。「2214」とか4文字の文字列。ハードに整合する値をプログラムで指定する。
4	コマンド		1bytes	byte	1バイト数値。
5	チャンネル番号		1bytes	byte	チャンネル番号 1〜8
6	接続情報		1bytes	byte	
7	10k値		2bytes	short	0 〜129 または 0〜257 不明あるいは適用外は -1。
8	100k値		2bytes	short	0 〜129 または 0〜257 不明あるいは適用外は -1。
9	最大値		4bytes		バランス時のみ
10	最小値		4bytes		バランス時のみ
11	バランス最終値		4bytes		バランス時のみ

4. コマンド

0x00 : 読み出し。MCPから現在の設定値を読み出してその値を返す。
0x01 : 書き込み。返される値は設定値した値で、ほんとに設定されたかどうかは未確認。
0x02 : バランス。返される値は最終値（現在の設定値。）

6. 接続情報

10kA, 10kW, 10kB, 100kA, 100kW, 100kB を示すビット。

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
10k valid	10kA	10kW	10kB	100k valid	100kA	100kW	100kB
bit 6〜4のデータが	接続していれば1、していなければ0			bit 2〜0のデータが	接続していれば1、していなければ0		

正しければ1、正しくなければ0		正しければ1、正しくなければ0	

hyg01：温度と湿度

プログラムを確認すると、テキスト (json?)で
humidity:??

gyr01

概要　ジャイロデータMPU6050を想定。

内容

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	前のUNIX時刻	ミリ秒	8byte	long	前回送信したユニット時刻
3	この後のデータバイト数	バイト	4byte	int	下記3と4の合計バイト数
4	データ個数	個	4byte	int	必要数、繰返し。
5	加速度	※1	データ個数 ×3byte	※2	

aht25：温度と湿度

概要

温湿度計の AHT25を想定。
温度と湿度を返す。

内容

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	温度	度	4byte	float	
3	湿度	%	4byte	float	

sht31：温度と湿度

概要

温湿度計の SHT25を想定。
温度と湿度を返す。

内容

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	データサイズ	個	4byte	int	
3	温度	度	4byte	float	
4	湿度	%	4byte	float	

ill01：照度

概要

照度センサのTSL2572を想定。

照度の配列を返す。

内容

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	データサイズ	個	4byte	int	ビッグエンディアン
3	照度	lx	2byte×データサイズ	uint16	ビッグエンディアン

memo: 覚書

	項目	単位	サイズ	型	
1	UNIX時刻	ミリ秒	8byte	long	送信直前のユニット時刻
2	このあとのデータバイト数	バイト	4byte	int	(4+内容のバイト数)
3	通し番号		4byte	int	
4	内容		不定	string	