MQTT and kdb+ on a Raspberry Pi

MQTT is an Internet of Things connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.

Kx have recently released an MQTT interface. Documented on code.kx.com with source code available on Github. The interface supports Linux/MacOx/Windows platforms.

Linux ARM 32-bit is also supported and that what this blog will use as an example running on an Raspberry Pi 3 Model B+.

Install and test a broker

An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. Eclipse Mosquitto is one of many implementations. We will use it for our example.

Install the broker with:

```
sudo apt-get install mosquitto
```

To test the broker install the command like tools:

```
sudo apt-get install mosquitto-clients
```

In one shell subscribe to messages on the `test` topic:

```
mosquitto_sub -h localhost -t test
```

In another shell publish a message on the same topic:

```
mosquitto_pub -h localhost -t test -m "hello"
```

You should see `hello` print to the subscriber shell.

Install kdb+

Download the 32-bit version of kdb+ and follow the install instructions. Making sure to rename `l32arm` to `l32` during install:

```
unzip linuxarm.zip
mv q/l32arm q/l32
mv q ~/
```

Ensuring to add `QHOME` to your `.bashrc`:

```
export PATH="$PATH:/home/pi/q/l32"
export QHOME="/home/pi/q"
```

Install the Kx MQTT interface

Full instructions for all platforms are on Github. Other platforms have fewer steps as there are pre compiled releases. For ARM will will compile from source.

Install needed dependencies which are needed to compile the projects:

```
sudo apt-get install libssl-dev cmake
```

The Paho MQTT C client library needs to be available before the kdb+ can be compiled. Take the link to the latest source code for the `paho.mqtt.c` library from it's releases tab on Github.

Download, compile and install:

```
wget https://github.com/eclipse/paho.mqtt.c/archive/v1.3.2.tar.gz
tar -xzf v1.3.2.tar.gz
cd paho.mqtt.c-1.3.2
make
sudo make install
cd
```

Now the system is ready for the Kx MQTT interface to be installed. Use it's releases tab on Github to find a link to the latest available.

```
wget -O mqtt.tar.gz https://github.com/KxSystems/mqtt/archive/1.0.0-rc.1.tar.gz
tar -xzf mqtt.tar.gz
cd mqtt-1.0.0-rc.1
mkdir cmake && cd cmake
cmake ..
make
make install
cd ..
cp q/mqtt.q $QHOME/
cp cmake/mqtt.so $QHOME/l32/
```

Test the Kx MQTT interface

Start a q process and subscribe to the `test` topic:

```
q)\l mqtt.q /Load the interface
q).mqtt.conn[`localhost:1883;`src;()!()] /Connect to the broker
q).mqtt.sub[`test] /Subscribe to the test topic
```

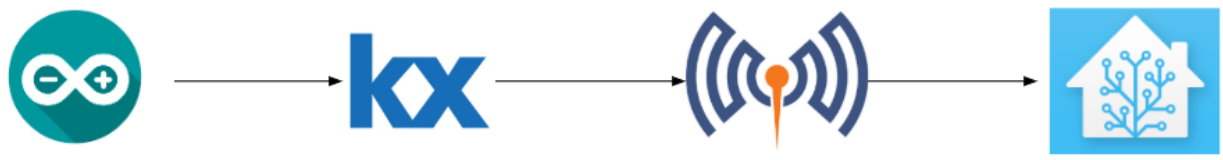
Publish a message. The default message receive function will print the incoming message:

```
q).mqtt.pub[`test;"hello"] /Publish to the test topic
2
q)(`msgsent;2)
(`msgrecvd;"test";"hello")
```

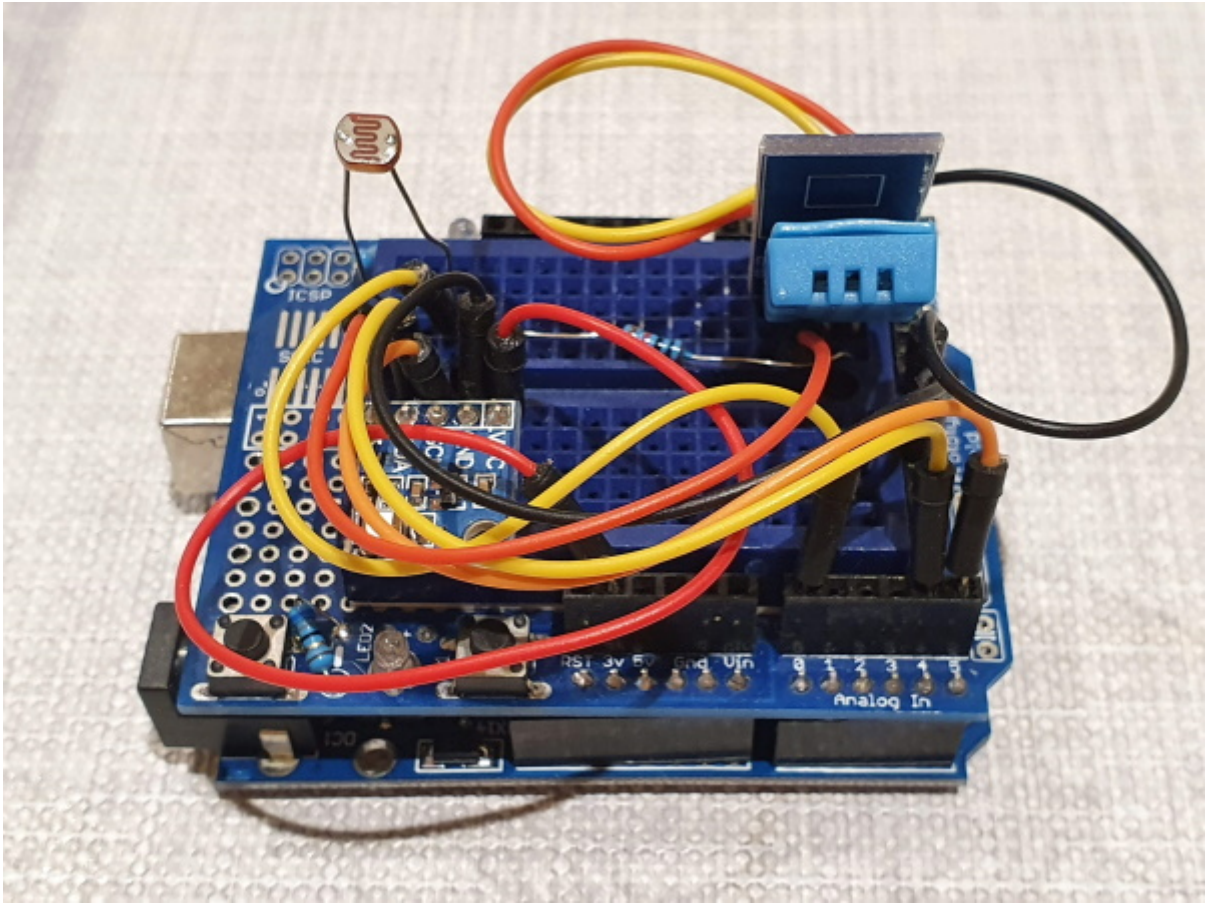
More examples are included on code.kx.com

Example use-case: Environmental Monitor

For a useful example we will collect some sensor data and publish it to an IoT platform. This blog will focus on the kdb+ aspects, full information and code is available on Github



The data source will be an Arduino microcontroller. Temperature, pressure, humidity and light readings will be read by kdb+ over a serial USB connection.



Read serial data with kdb+

Reading the serial data in kdb+ is quick using named pipe support:

```
q)ser:hopen`$":fifo://",COM
q)read0 ser
"26.70,35,736,1013,-5.91,26421"
```

The comma seperated field contain:

1. Temperature - Celsius
2. Humidity - Percent
3. Light - Analog value between 0 and 1023
4. Pressure - Pa
5. Altitude - m (Not accurate)
6. CRC-16 - checksum of data fields

Calculate an error detecting checksum in kdb+

The final field is particularly important. This is a checksum which enables error-detection, a requirement as serial data can be unreliable. Without this incorrect data could be interpreted as correct. For example a temperature reading such as `26.70` missing it's decimal point would be published as `2679`.

In kdb+ a function is needed to generate a checksum to compare against the one sent by the Arduino. If the two values do not match the data is rejected as it contains an error.

To create the function the logic from C functions crc16_update and calcCRC was created as `crc16`. The over accumulator was used in place of `for` loops:

```
rs:{0b sv y xprev 0b vs x} /Right shift
ls:{0b sv neg[y] xprev 0b vs x} /Left shift
xor:{0b sv (<>/)vs[0b] each(x;y)} /XOR
land:{0b sv (.q.and). vs[0b] each(x;y)} /Logical AND

crc16:{
 crc:0;
 {x:xor[x;y];
  {[x;y] $[(land[x;1])>0;xor[rs[x;1];40961];rs[x;1]]} over x,til 8
 } over crc,`long$x
 };
```

In this example pressure can be seen to have arrived incorrectly as `101020` rather than `1020`:

```
Failed checksum check
Error with data
26.30,36,739,101020,-56.88,17352
```

Publish the data to Home Assistant

Home Assistant is a home automation platform. To make the sensor data captured available to Home Assistant it can be published to MQTT.

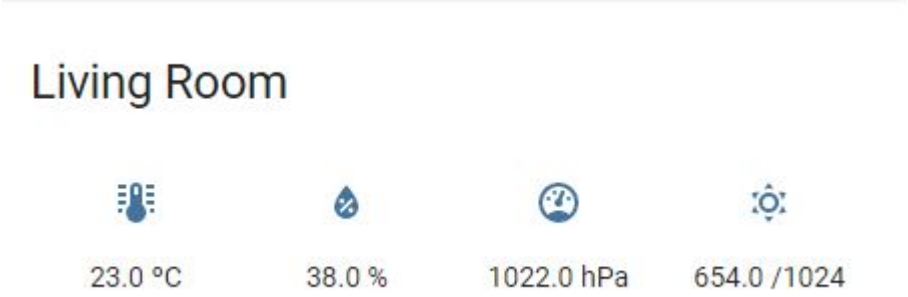
Every 5 seconds data is read from the serial port. It's checksum is validated. If the data is without error the data is published to the MQTT broker.

```
port:1883
broker_address:.z.x[0]
COM:.z.x[1]
room:.z.x[2]

.mqtt.conn[`$broker_address,":",string port;`src;()!()]

ser:hopen`$":fifo://",COM

pub:{[]
 rawdata:last read0 ser;
 @[{
    qCRC:crc16 #[;x] last where x=",";
    data:"," vs x;
    arduinoCRC:"J"$last data;
    if[not qCRC=arduinoCRC;'"Failed checksum check"];
    .mqtt.pub[`$"hassio/",room,"/temperature";data[0]];
    .mqtt.pub[`$"hassio/",room,"/humidity";data[1]];
    .mqtt.pub[`$"hassio/",room,"/light";data[2]];
```

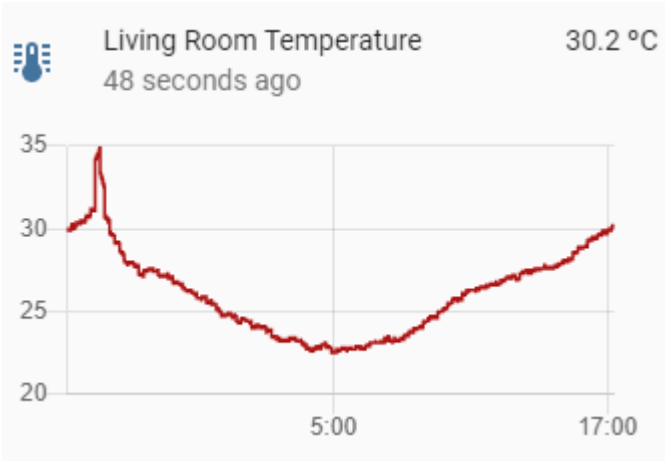
```
      .mqtt.pub[`$"hassio/",room,"/pressure";data[3]]
    };
    rawdata;
    {-1 "Error with data: \"",x,"\" '",y}[rawdata]
   ];
  };

.z.ts:{pub[]}

\t 5000
```

Home Assistant then needs only some configuration entries which will subscribe to the data and make it available for automate as well as display through it's UI.

An overview of all the sensors:



Clicking on any one sensor allows a more detailed graph to be seen:



Storing MQTT sensor data in Kdb+

Home Assistant displays 24hours of data. A long term store of sensor data is valuable for trend analysis.

Subscribing to the published data from another kdb+ process is quick:

```
q)\l mqtt.q
q).mqtt.conn[`localhost:1883;`src;()!()]
q).mqtt.sub[`$"hassio/livingroom/#"]
```

The topic information can be seen arriving with the data:

```
(`msgrecvd;"hassio/livingroom/temperature";"22.7")
(`msgrecvd;"hassio/livingroom/humidity";"38.0")
(`msgrecvd;"hassio/livingroom/light";"582.0")
(`msgrecvd;"hassio/livingroom/pressure";"1018.0")
```

Prepare a table to store this data:

```
readings:([] time:`timestamp$();room:`$();sensor:`$();reading:`float$())
```

Customise the message receiver to insert the data to the table:

```
.mqtt.msgrcvd:{`readings insert .z.p,@[`$"/" vs x;1 2],"F"$y}
```

Data is now stored in memory:

```
q)readings
time                          room       sensor      reading
-------------------------------------------------------------
2020.05.23D07:58:34.789984000 livingroom temperature 22.9
2020.05.23D07:58:34.790654000 livingroom humidity    38
2020.05.23D07:58:34.791065000 livingroom light       644
2020.05.23D07:58:34.791442000 livingroom pressure    1018
```

Extending again we can save data to disk daily. The historic data table is named `readgingsHist`, this is done to keep the example simple. Both historic and real-time data can be queried within one process.

```
day:.z.d
HDB:`:/home/pi/sensorHDB
.z.zd:17 2 6

.mqtt.msgrcvd:{
    now:.z.p;
    if[day<`date$now;
       .Q.dd[HDB;(`$string day;`readgingsHist;`)] set
.Q.ens[HDB;readings;`sensors];
       `readings set 0#readings;
       day:`date$now;
       system"l ",string HDB;
     ];
    `readings insert .z.p,@[`$"/" vs x;1 2],"F"$y
    }
```

ToDo:

1. Update error format
2. Add Dashboards install instructions
3. Add analytics
4. Create historical dashboard