HAPI Docs Documentation

Release 0.0

HAPI

Jul 12, 2017

Contents

The Hydoponic Automation Platform Initiative (HAPI) is a suite of tools that will allow people to grow a variety of food in diverse environments. Our primary focus is on building intelligent automation for hydroponics and aquaponics. HAPI is currently under heavy development.

To get started with HAPI check out the *System Overview*. You can find out more about the project on the website.

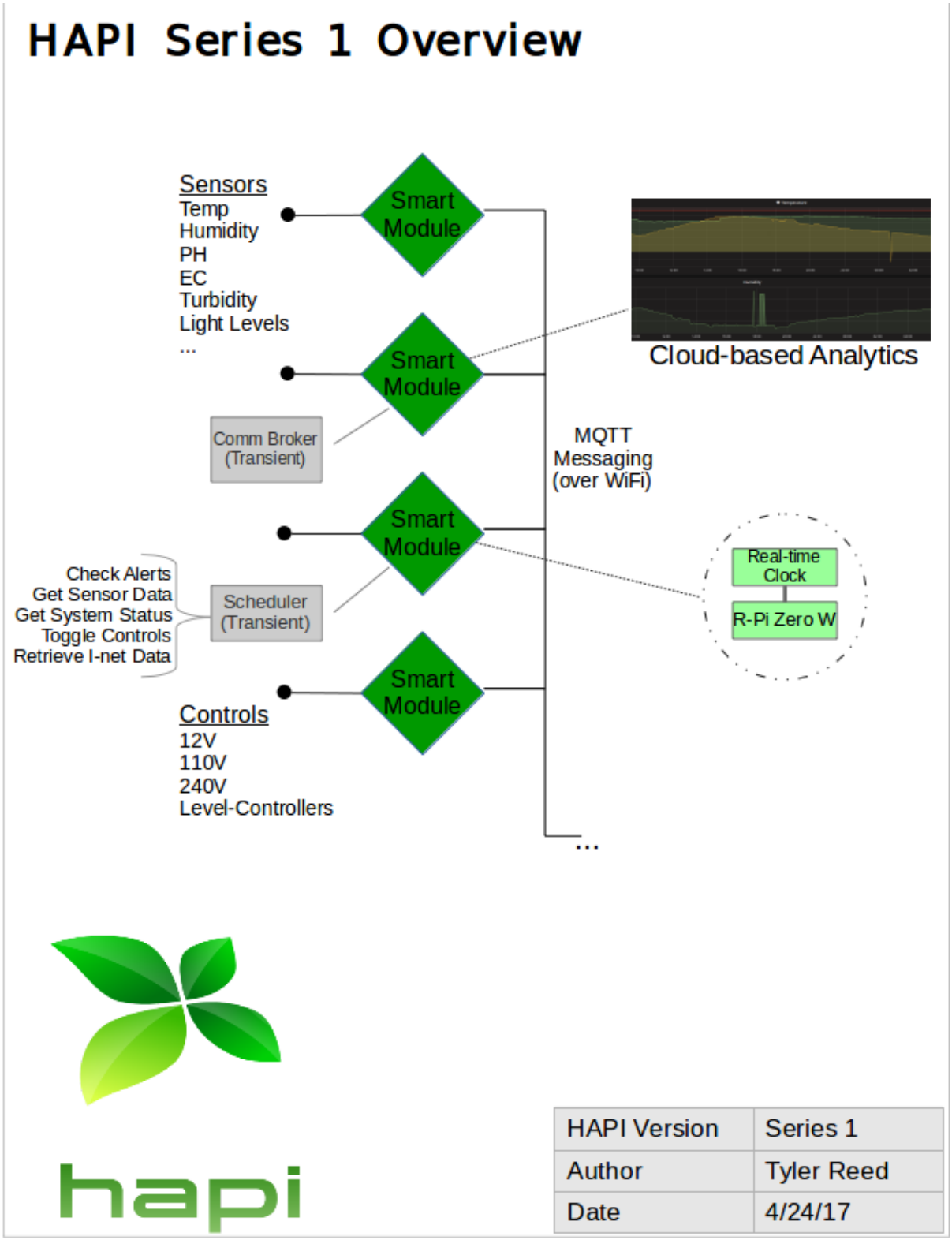
Contents:

CHAPTER 1

System Overview

The HAPI system is based on low cost micro-controllers providing sufficient computing power to monitor and control all local production resources. These resources are comprised of the sensors that monitor the environment and growing conditions, and the controls that affect these conditions. The results of these activities are published using a standard messaging protocol called MQTT. The information that is published can then be used to analyze the conditions, monitor the controls, and change the operation of the system.

Below is a draft of the HAPI system architecture. Please note it is not fully up to date


HAPI Series 1 Overview
Sensors
Temp
Humidity
PH
EC
Turbidity
Light Levels
...
Smart Module
Smart Module
Smart Module
Smart Module
Comm Broker (Transient)
Scheduler (Transient)
Check Alerts
Get Sensor Data
Get System Status
Toggle Controls
Retrieve I-net Data
Controls
12V
110V
240V
Level-Controllers
MQTT Messaging (over WiFi)
Cloud-based Analytics
Real-time Clock
R-Pi Zero W
...
hapi
HAPI Version
Series 1
Author
Tyler Reed
Date
4/24/17

Components

HAPImodule

HAPImodules are devices with a high computational capability that run system, sensor and control code. Currently the supported hardware for Smart Modules is a Raspberry Pi Zero W or 3B. Multiple HAPImodules can exist within the system and collaborate to determine the tasks performed by each module. One HAPImodule is the MQTT Broker (Server) and is responsible for storing all the data being transmitted. If for some reason this Server is unreachable (configuration problems or hardware failure, for instance), any other HAPImodule within the system can step up and take its place.

Features:

- Focused on the needs of Urban and Controlled-Environment Agriculture
- Supports multiple sites for a single user
- Open source designs and code (user isn't stuck with proprietary technology)
- Wide & growing range of sensor support
- User-definable Alerts
- User-definable Schedule
- Control equipment based on schedule, sensor values or manually.
- WiFi Capable (configured via FMS Application)
- Seamlessly integrate with other Smart Modules
- Integrated, battery-backed, real-time clock
- Simple, durable design and enclosure
- Waterproof, weather-resistant enclosures
- Optional data push to cloud visualization services
- Automatic fail-over of communications and scheduling functions (fault-tolerant mesh)
- Minimal User Configuration (Set WiFi info and you're done)

HAPInode

The hardware for the HAPInode is based on arduino. Since the main focus of HAPI is a system wide automation, arduinos that can easily incorporate WiFi and Bluetooth Low Energy are preferred. Conditional defines in the arduino software allow for ESP32, ESP8266 and mega2560 w/Ethernet modules. The preferred development module is (https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide), which includes the necessary WiFi, Bluetooth, external power and external battery interfaces.

Features:

- Run on Arduino, ESP8266 (e.g. NodeMCU) or ESP32 (e.g. WROOM32)
- Targeted to DIY/Makers, Technicians, Experimenters, Hobbyists
- User-configurable sensor and control configurations
- Seamlessly integrate with other HAPI Smart Modules and Nodes
- Open source designs and code

Facility Management System (FMS) Android App

The FMS is an Android application for managing and monitoring the HAPI system.

- Configure WiFi settings for Smart Modules
- See all sensor values at a touch
- Manage schedule for collecting sensor data, weather data, system health and checking alerts
- Set Alert parameters and notification preferences
- View data in Standard or Metric Units
- Open source designs and code
- Integration with Weather Underground (optional via free subscription)
- Access to Visualization Dashboards (optional via paid subscription)

System Features

Job scheduling

The HAPI platform features an in-process scheduler for running periodic jobs. Jobs can be control functions, such as turning a light or pump on. Jobs can also serve monitoring purposes, such as gathering sensor data. Job information is stored in the database, in the table *schedule*. Any Smart Module can run the job scheduling code (i.e. "become the Scheduler"). HAPI Nodes cannot become the Scheduler.

Technical Description

When a Smart Module starts, it publishes on message on the MQTT topic SCHEDULER/QUERY, essentially asking for the Scheduler to respond. If another module on the network is running the job scheduling code, it responds to the to the message and no further action is taken. If there is no response to the message, the starting module runs the job scheduling code itself, announces that it is now the Scheduler and listens on the SCHEDULER/QUERY topic for other Smart Modules that are discovering the scheduler.

The HAPI Scheduler uses [the schedule package](https://pypi.python.org/pypi/schedule) created by Daniel Badr. Documentation can be [found here](https://schedule.readthedocs.io/en/stable/).

Auto-Discovery

All modules (Smart or Dumb) have the capability to automatically detect and connect to one another without user configuration or intervention. To accomplish this, the HAPI platform uses a zero-configuration networking implementation called Avahi. Similar to Apple's Bonjour, Avahi allows modules to dynamically name themselves and discover neighboring modules.

Technical Description

For Smart Modules, the original Raspian images are modified to include the avahi daemon. When a module boots, it automatically runs this daemon. When a Smart Module starts, to attempts to contact the MQTT broker, "mqttbroker.local". This machine is acts as the communications hub for all HAPI-based facilities. If the Smart Module's search for the MQTT broker is not successful, it changes it's own name to "mqttbroker.local" and becomes the communications hub for the site.

CHAPTER 2

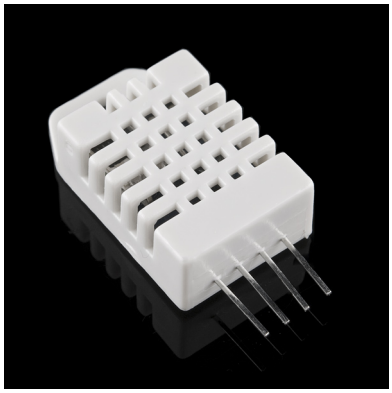
HAPImodule

Release 1 of the HAPI system uses Raspberry Pi Zeros (HAPiZ) as the **Smart Modules** that run the system code, the sensor code and the control code. Multiple HAPiZ devices can exist within the system and collaborate to determine the tasks performed by each module.

Hardware Setup

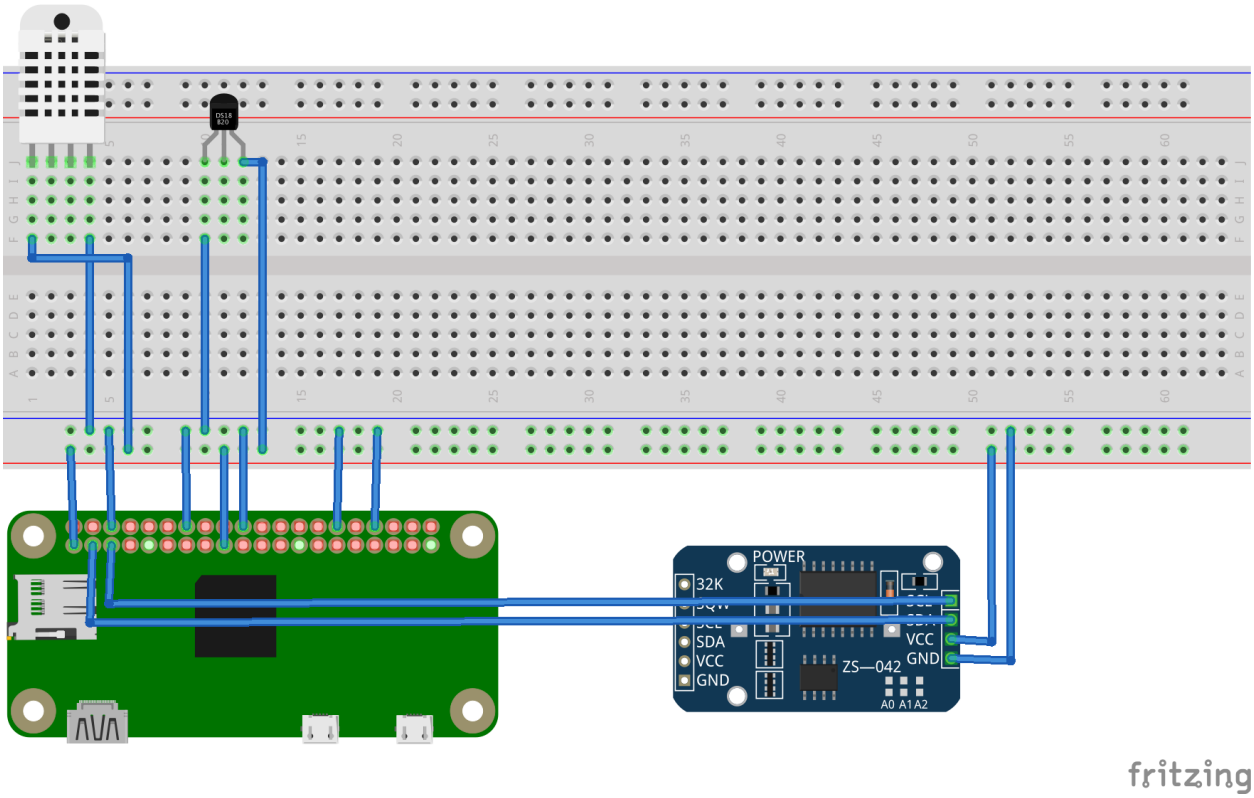
A minimum HAPiZ module consists of the following components:

- Raspberry Pi Zero
- Real-time-clock (DS3231)
- Temperature and humidity sensor (DHT22)



- Water temperature sensor (DS18B20)

Prototype Diagram



Software Setup

Note: Soon we'll introduce configuration via regular file and/or database. For now all configuration are hardcoded.

1. Install Raspbian on the Raspberry Pi

- Raspbian installation guide.
- **Configure Raspbian Jessie**
  - At minimum, using ssh or graphical shell:

```
sudo raspi-config
```

  - Set password
  - Set hostname (e.g.HAPImodule001)
  - Set locale
  - Set keyboard
  - Set timezone
  - Set wifi - country, ssid, and password
  - Reboot

2. Install dependencies on the Pi

- Dependencies:
  - **Avahi** (daemon) configured to publish MQTT service. avahi is already installed on Raspi Jessie.
    * `Example config`
    * Copy the file to the config directory and rename it:

```
sudo cp ~/Downloads/avahi-example /etc/avahi/services/multiple.service
```

    * Restart to apply changes:

```
systemctl restart avahi-daemon.service
```

  - **MQTT Mosquitto** with default configuration.
    * Install mosquitto

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

    * `Example config`
    * Copy the file to the config directory and rename it:

```
sudo cp ~/Downloads/mosquitto-example /etc/mosquitto/conf.d/mosquitto.conf
```

    * Start mosquitto to apply config, and set mosquitto to start on boot:

```
sudo systemctl start mosquitto
sudo systemctl enable mosquitto
```

  - **Influxdb with default configuration.** *https://easysquirrel.io/index.php/2017/03/20/influxdb-and-telegraf-on-raspberry-pi-3/ http://docs.influxdata.com/influxdb/v1.2/introduction/installation*
    * the configuration file is located at /etc/influxdb/influxdb.conf for default installations
    * test by typing 'influx' at the command line
    * **add influxdb repository**

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/os-release
test $VERSION_ID = "7" && echo "deb https://repos.influxdata.com/debian␣
↪wheezy stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
test $VERSION_ID = "8" && echo "deb https://repos.influxdata.com/debian␣
↪jessie stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
```

```
```

* Install libfontconfig1 (required)

```
sudo apt-get install libfontconfig1
sudo apt-get -f install
```

* **influxdb**

```
sudo apt-get update && sudo apt-get install influxdb
sudo service influxdb start
```

* **Telegraf**

```
sudo apt-get update && sudo apt-get install telegraf
sudo service telegraf start
```

- **Grafana** [Optional] (highly recommended)

```
cd ~
wget --output-document=grafana_4.2.0-beta1_armhf.deb https://bintray.
↪com/fg2it/deb/download_file?file_path=testing%2Fg%2Fgrafana_4.2.0-
↪beta1_armhf.deb
sudo dpkg -i grafana_4.2.0-beta1_armhf.deb
sudo apt-get install -f
```

- Enable Grafana for automatic start on boot and start the server

```
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

- Reboot your Raspberry Pi

```
sudo reboot
```

3. Install hapi

- **Install hapi dependencies on the Pi** * There is also a script from the repo for installing the first two system dependencies and then setup the python environment:

```
sudo apt-get install git
cd ~
git clone https://github.com/mayaculpa/hapi.git
cd ~/hapi/src/smart_module
./INSTALL.sh
```

- It is good practice to look over scripts you download from the Internet before running them.

Usage

Start the program:

```
python smart_module.py
```

You should get output like this:

```
(venv) $ python smart_module.py
2017-07-02 12:15:58.202878 - smartmodule.log - [*] INFO - Communicator initialized
Mock Smart Module hosting asset HSM-WT123-MOCK wt Environment.
2017-07-02 12:15:58.211355 - smartmodule.log - [*] INFO - Performing Broker discovery.
↪..
2017-07-02 12:16:01.213817 - smartmodule.log - [*] INFO - MQTT Broker: ArchMain.local.
↪ IP: 192.168.0.99.
2017-07-02 12:16:04.217127 - smartmodule.log - [*] INFO - Connecting to ArchMain.
↪local. at 192.168.0.99.
2017-07-02 12:16:04.218420 - smartmodule.log - [*] INFO - Closing Zeroconf connection.
2017-07-02 12:16:04.239513 - smartmodule.log - [*] INFO - Connected with result code 0
$SYS/broker/clients/total 0
2017-07-02 12:16:08.720840 - smartmodule.log - [*] INFO - No Scheduler found.␣
↪Becoming the Scheduler.
2017-07-02 12:16:08.721437 - smartmodule.log - [*] INFO - Loading Schedule Data...
2017-07-02 12:16:08.748795 - smartmodule.log - [*] INFO - Schedule Data Loaded.
2017-07-02 12:16:08.749374 - smartmodule.log - [*] INFO -   Loading seconds job:␣
↪System Status.
2017-07-02 12:16:08.749580 - smartmodule.log - [*] INFO -   Loading seconds job:␣
↪Check Alert.
2017-07-02 12:16:08.750986 - smartmodule.log - [*] INFO - Scheduler program loaded.
2017-07-02 12:16:08.753495 - smartmodule.log - [*] INFO - Influxdb information loaded.
2017-07-02 12:16:08.755970 - smartmodule.log - [*] INFO - Site data loaded.
$SYS/broker/clients/total 1
Running command self.smart_module.on_check_alert()
ASSET/QUERY Is it warm here?
ASSET/RESPONSE/HSM-WT123-MOCK {"value_current": "31.0", "name": "Temperature Sensor",
↪"context": "Environment", "virtual": 1, "type": "wt", "enabled": 1, "id": "HSM-
↪WT123-MOCK", "unit": "C", "system": ""}
2017-07-02 12:16:19.070110 - smartmodule.log - [*] INFO - Wrote to analytic database.
2017-07-02 12:16:19.070215 - smartmodule.log - [*] INFO - Fetching alert parameters␣
↪from database.
2017-07-02 12:16:19.070936 - smartmodule.log - [*] INFO - Closing Alert database␣
↪connection.
2017-07-02 12:16:19.071006 - smartmodule.log - [*] INFO - [!] ALERT DETECTED. Value:␣
↪31.0.
ALERT/HSM-WT123-MOCK {"upper": 30.0, "lower": 10.0, "value_current": "31.0", "response
↪": "email,sms", "message": "Houston, we have a problem", "notify_enabled": 1, "id":
↪"HSM-WT123-MOCK"}
2017-07-02 12:16:19.071630 - smartmodule.log - [*] INFO - Sending email notification.
2017-07-02 12:16:19.072025 - smartmodule.log - [*] INFO - Mail settings loaded.
2017-07-02 12:16:22.287407 - smartmodule.log - [*] INFO - Email notification sent.
2017-07-02 12:16:22.485832 - smartmodule.log - [*] INFO - Sending SMS notification.
[...]
```

An important note: we're currently using sqlite3 database to load schedule jobs and others information. You can check/use a demo of the database here: `database-example` For now you should place it on the same folder as *smart_module.py* and name it as *hapi_core.db*.

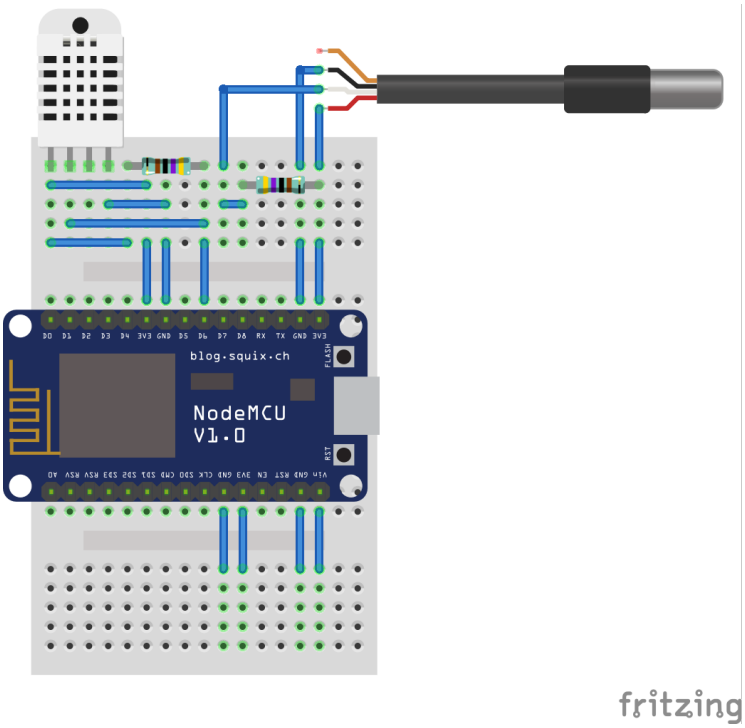
CHAPTER 3

HAPInode

Hardware Setup

The hardware for the HAPInode is based on arduino. Since the main focus of HAPI is a system wide automation, arduinos that can easily incorporate WiFi and Bluetooth Low Energy are preferred. Conditional defines in the arduino software allow for ESP32, ESP8266 and mega2560 w/Ethernet modules. The preferred development module is https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide, which includes the necessary WiFi, Bluetooth, external power and external battery interfaces.

Prototype Diagram



Software Setup

Using the Arduino IDE download and flash the board with the HAPInode code.

CHAPTER 4

MQTT Topics

In the HAPI system devices are either **modules** or **nodes**. Both types of devices can have sensors and controls attached to them. Modules are based on Raspberry Pis, while nodes are based on arduinos. The communication of sensor data or control commands uses the **MQTT protocol**. This protocol has two important operations called publishing and subscribing. In the HAPI system, a **local broker** is responsible for these operations.

The HAPI philosophy is that any module can assume the role of local broker. It then has the role of publishing controls based on a schedule, on sensor values or on exceptions, as well as co-ordinating the collection and storage of the sensor data.

A HAPI device can publish its sensor data and subscribe to controls for actions. A HAPI device could also subscribe to sensor data for processing and generate controls based on schedules or on these data values.

A HAPImodule can both publish and subscribe to data as well as publish and subscribe to controls. A HAPImodule can also assume the role of the MQTT local broker. A HAPInode can only publish data and subscribe to controls.

Device Naming

Each device has a unique ID, called the deviceID, derived from its type and the unique mac address that is hardcoded into the device at the time of manufacture. Each sensor or control is called an asset and has an assetID that is unique to the function of the asset. The deviceID and the assetID are used to uniquely identify the topic that sensor data is published to or that control information is received from.

HAPImodule

HAPImodules have a deviceID of the form HN1xxxxxx, where HN1 identifies that this is a module (Raspberry Pi Zero) and xxxxxx is the low three bytes of the mac address of its WiFi module. (The high three bytes identify the manufacturer of the WiFi integrated circuit). In python, the mac address is found using: *from uuid import getnode as get_mac mac = get_mac()*

HAPInode

Each node has a unique ID derived from its type (a node and its arduino type) and the unique mac address that is hardcoded into the device at the time of manufacture. HAPInodes have a name of the form HNnxxxxxx, where HNn identifies that this is a node (ESP32(HN5), ESP8266(HN4), or mega2560(HN3)) and xxxxxx is the low three bytes of the mac address of its WiFi module. (The high three bytes identify the manufacturer of the WiFi integrated circuit). In arduino with a WiFi module, the mac address is found using: ‘mac = WiFi(mac); ‘

MQTT topics

Each topic is built from the nature of the topic, the activity to be undertaken, and optionally, the deviceID and the assetID. The first part is the nature of the topic, e.g. STATUS, ASSET, EXCEPTION, CONFIG The next part is the activity associated with that topic, e.g. QUERY, RESPONSE, SET, CLEAR

Sample topics

Status Query

Query the status of ALL devices topic: STATUS/QUERY/ message: {“device”: “*”}

Status Response

topic: STATUS/RESPONSE/HN1123456 Message contains JSON encoded fields identifying the device, name, and assets. message: {“device”: “HN1123456”, “name”: “TomatoBay1”, “Asset”: “[hum, tmp, wtm, lux]”} Note that multiple messages may be generated to identify all the assets associated with a device, as the maximum MQTT payload length is limited to approximately 96 bytes, or 128 byte message length.

Asset Query

Query the value of the hum (humidity) asset of all devices with humidity assets topic: ASSET/QUERY/+/hum message: {“device”: “*”}

Asset Response

topic: ASSET/RESPONSE/HN1123456/hum Message contains JSON encoded fields identifying the device, time, asset value, and units. message: {“device”: “HN1123456”, “time”: 123456, “hum”: 67, “units”: “%”}