AurAir
RESTful API Guide

1 Executive Summary

The AurAir Base firmware offers a RESTful API to read, update and insert values from the AurAir Base. Controlled by it's database, the AurAir Base RESTful API is exposing the database so that it's possible to directly read values from the AurAir and change it's behaviour.

USE AT OWN RISK!

CHANGING AURAIR BASE DATA USING THE RESTFUL API MIGHT RESULT IN A NON-FUNCTIONAL AURAIR BASE!

IF THE AURAIR IS NO LONGER FUNCTIONING, A FACTORY RESET MIGHT BE RESTORING IT'S FUNCTIONALITY.

AurAir
RESTful API Guide

2 Version History

Date	Version	Comments
11-5-2019	1.0	Initial draft
12-11-2019	1.1	Cloud/wifi/password/locked config added, humidity added

3 CONTENT

4 Purpose

The AurAir Base firmware can be used to turn the device into an easy multifunction sensor device for Home Automation solutions like Domoticz using this RESTful API. Configuration of AurAir Base is entirely web based, so once you've got the firmware loaded, you don't need any other tool besides a common web browser.

Due to it's rapid Agile development cycles, is AurAir Base primarily developed for the ESP32 platform.

5 Scope

The scope of AurAir Base:

- Models B1 or B2

6 REST API Architecture Principles

REST API Architecture principles are used to steer the development of AurAir Base so that the development doesn't get make solutions, or get lost in solving problems, that are not needed.

The principles[1] are:

- performance in component interactions, which can be the dominant factor in user-perceived performance and network efficiency.
- scalability allowing to support large numbers of components and interactions among components. Roy Fielding describes REST's effect on scalability as follows:
    - REST's client–server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.
- simplicity of a uniform interface.
- modifiability of components to meet changing needs (even while the application is running).
- visibility of communication between components by service agents.
- portability of components by moving program code with the data..
- reliability in the resistance to failure at the system level in the presence of failures within components, connectors, or data.

[1] Wikipedia 2018: https://en.wikipedia.org/wiki/Representational_state_transfer

7 Architecture Constraints

Six guiding constraints define a RESTful system.These constraints restrict the ways that the server may process and respond to client requests so that, by operating within these constraints, the service gains desirable non-functional properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

The formal REST constraints [2]are as follows:

- **Client–server architecture**

The principle behind the client–server constraints is the separation of concerns. Separating the user interface concerns from the data storage concerns improves the portability of the user interface across multiple platforms. It also improves scalability by simplifying the server components. Perhaps most significant to the Web, however, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains.

- **Statelessness**

The client–server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client. The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time the client chooses to initiate a new state-transition.

- **Cacheability**

As on the World Wide Web, clients and intermediaries can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

- **Layered system**

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies.

- **Code on demand (optional)**

Servers can temporarily extend or customize the functionality of a client by transferring executable code. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

[2] Wikipedia 2018: https://en.wikipedia.org/wiki/Representational_state_transfer

- **Uniform interface**

The uniform interface constraint is fundamental to the design of any REST service.It simplifies and decouples the architecture, which enables each part to evolve independently. The four constraints for this uniform interface are:

- **Resource identification in requests**

Individual resources are identified in requests, for example using URIs in Web-based REST systems. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server may send data from its database as HTML, XML or JSON, none of which are the server's internal representation.

- **Resource manipulation through representations**

When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.

- **Self-descriptive messages**

Each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by a media type.

- **Hypermedia as the engine of application state (HATEOAS)**

Having accessed an initial URI for the REST application—analogous to a human Web user accessing the home page of a website—a REST client should then be able to use server-provided links dynamically to discover all the available actions and resources it needs. As access proceeds, the server responds with text that includes hyperlinks to other actions that are currently available. There is no need for the client to be hard-coded with information regarding the structure or dynamics of the REST service.

8 Baseline Architecture

8.1 HTTP Response Codes

Another important part of REST is responding with the correct status code for the type of request that was made. When you make an HTTP request, the server will respond with a code which corresponds to whether or not the request was successful and how the client should proceed. There are four different levels of codes:

2xx = Success

3xx = Redirect

4xx = User error

5xx = Server error

Here's a list of the most important status codes:

Success codes:

200 - OK (the default)

201 - Created

202 - Accepted (often used for delete requests)

User error codes:

400 - Bad Request (generic user error/bad data)

401 - Unauthorized (this area requires you to log in)

404 - Not Found (bad URL)

405 - Method Not Allowed (wrong HTTP method)

409 - Conflict (i.e. trying to create the same resource with a PUT request)

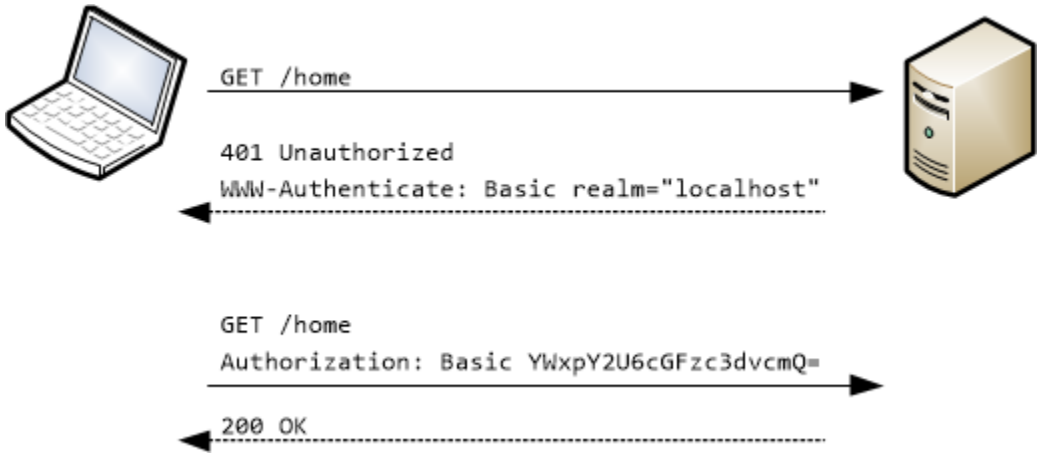
8.2 HTTP Response Body

The HTTP GET request is answered with an HTTP Response and body, the body contains JSON encoded or plain text data. HTTP PUT, POST and DELETE requests require an HTTP body containing JSON encoded data.

8.3 Authentication and Authorization

8.3.1 Authentication

The most simple way to deal with authentication is to use HTTP basic authentication. We use a special HTTP header where we add 'username:password' encoded in base64.



Digest authentication is not supported in v1.0

8.3.2 Authorization

The username is standard ‘admin’, multi user support is not included in v1.0, the admin user is authorized for all services.

8.4 Base URL

The base url for the RESTful API is:

HTTP://<AurAir Base ip or name>:<ip port>/api/vx.x/

Where vx.x stands for version x.x, for example v1.0

x.x is minimal 1.0 and each new API definition will use a higher number then the base number (like v1.1). The standard <ip port> is 80.

Example url: **http://aurairbase/api/v1.0/system**

8.5 Resources

For each of the elements is the interaction listed for GET, POST, PUT and DELETE HTTP requests. If one or more of the HTTP requests are not listed for the element, then the element does not support this HTTP request.

8.5.1 SYSTEM

Methods:	[GET]
URI:	<base url>/system
Body:	No body or headers allowed
Actions:	Retrieve basic information of the AurAir Base configuration.
Firmware version:	1.04

JSON response data example:

{
"dbversion": "Prod",
"holder": "AurAir",
"free": 31184,
"uptime": "0 days 4 hours 14 minutes",
"current": -1,
"unit": 0,
"wifimode": 4,
"name": "AurAir",
"stack": 2608,
"total": 111168,


```
 "freq": "-",
 "copyright": "Copyright 2019",
 "time": "2019-08-27 12:23:30",
 "unique": "b1-d8a01d589320",
 "buildversion": "Prod",
 "srcname": "AurAir",
 "version": "1.0.3",
 "peak": -1,
 "buildstatus": "Prod"
}
```


8.5.2 REBOOT

Methods: GET]

URI: <base url>/system/reboot

Body: No body or headers allowed

Actions: Reboot the AurAir Base with a 3 second delay.

Firmware version: 1.04

Plain text data example:

System rebooting!

8.5.3 REBOOTNOW

Methods: GET]

URI: <base url>/system/rebootnow

Body: No body or headers allowed

Actions: Reboot the AurAir Base immediately.

Firmware version: 1.05

Plain text data example:

(none)

8.5.4 FACTORY RESET

Methods: [GET]

URI: <base url>/system/factory

Body: No body or headers allowed

Actions: Factory reset to the default AurAir Base configuration. A reboot is needed to make this effective.

Firmware version: 1.04

Plain text data example:

Factory reset done!

8.5.5 FIRMWARE UPDATE

Methods: [GET]

URI: <base url>/system/firmware

Body: No body or headers allowed

Actions: Update the AurAir Base firmware to the latest available version. It will reboot the AurAir to the firmware update tool.

Firmware version: 1.04

Plain text data example:

Updating firmware….

In case you do NOT have an active internet connection, the following message appears:

Updating firmware NOT possible, not connected to Internet!

8.5.6 WIFISCAN

Methods: [GET]

URI: <base url>/system/wifiscan

Body: No body or headers allowed

Actions: List all WiFi networks available to the AurAir Base.

Firmware version: 1.04

JSON data example:


```
[
 {
  "strength": "1",
  "security": "Open",
  "hidden": "Visible",
  "ssid": "AccessPoint",
  "channel": "1"
 },
 {
  "strength": "-65",
  "security": "WPA2-PSK",
  "hidden": "Visible",
  "ssid": "Ziggo",
  "channel": "11"
 }
]
```


8.5.7 I2CSCAN

Methods: [GET]

URI: <base url>/system/i2cscan

Body: No body or headers allowed

Actions: List all I2C devices bus id's available to the AurAir Base.

Firmware version: 1.04

JSON data example:


```
{
 0: 24,
 1: 118
}
```


8.5.8 CONFIG

Methods: [GET,PUT]

URI: <base url>/config

Body: GET: No body or headers allowed

PUT: JSON Body containing only changed data

Actions: List or update AurAir Base configuration data. A reboot is needed to make this effective.

Firmware version: 1.04

JSON data example:


```
{
 "xs2": "",
 "xs3": "",
 "cloudpassword": "SKAfNzI1MTbzNw",
 "xs1": "",
 "port": 80,
 "unit": 0,
 "sleeptime": 60,
 "name": "AurAir",
 "xi1": 0,
 "timestamp": "2000-01-01_00-00-091038899934",
 "xi2": 0,
 "opaque": "OJP3MEY4STEw",
 "ssl": "off",
 "sleepfailure": "",
 "password": "",
 "sleepenable": "",
 "nonce": "WNA8MykzWDAz7m",
 "xi3": 0,
 "version": "Prod",
 "locked": 0
}
```


8.5.9 CLOUD CONFIG

Methods: [GET,PUT]

URI: <base url>/config/cloud

Body: GET: No body or headers allowed

PUT: JSON Body containing on/off data, switching it on or off.

Actions: List or update AurAir Base cloud configuration data. A reboot is needed to make this effective.

Firmware version: 1.05

JSON data example:

```
{
  "cloud": "on"
}
```

8.5.10 WIFI CONFIG

Methods: [GET,PUT]

URI: <base url>/config/wifi

Body: GET: No body or headers allowed

PUT: JSON Body containing ssid and key (password) data.

Actions: List or update AurAir Base wifi configuration data. A reboot is needed to make this effective.

Firmware version: 1.05

JSON data example:

```
{
  "ssid": "AccesPoint1",
  "key": "YourPassword"
}
```

8.5.11 PASSWORD CONFIG

Methods: [PUT]

URI: <base url>/config/password

Body: PUT: JSON Body containing administrator password data.

Actions: List or update AurAir Base administrator password. A change is effective immediately.

Firmware version: 1.05

JSON data example:

```
{
  "password": "test"
}
```

8.5.12 LOCKED CONFIG

Methods: [GET,PUT]

URI: <base url>/config/locked

Body: GET: No body or headers allowed

PUT: JSON Body containing locked on/off data, switching firmware update/calibration/factory reset/5m password button functions on or off.

Actions: List or update AurAir Base button configuration data. A reboot is needed to make this effective.

Firmware version: 1.05

JSON data example:

```
{
  "locked": "on"
}
```

8.5.13 NAME CONFIG

Methods: [GET,PUT]

URI: <base url>/config/name

Body: GET: No body or headers allowed

	PUT: JSON Body containing AurAir Base unit name.
Actions:	List or update AurAir Base unit name configuration data.
Firmware version:	1.05

JSON data example:

```
{
  "name": "test"
}
```

8.5.14 LOCATION CONFIG

Methods:	[GET,PUT]
URI:	<base url>/config/location
Body:	GET: No body or headers allowed
	PUT: JSON Body containing AurAir Base unit location.
Actions:	List or update AurAir Base unit location configuration data.
Firmware version:	1.06

JSON data example:

```
{
  "location": "room1"
}
```

8.5.15 NETWORK

Methods:	[GET, PUT]
URI:	<base url>/network
Body:	GET: No body or headers allowed
	PUT: JSON Body containing only changed data
Actions:	List or update AurAir Base network data. A reboot is needed to make a change effective.
Firmware version:	1.04

JSON data example:


```
{
 "xs2": "",
 "cs": "",
 "ip": "",
 "fbssid": "",
 "timestamp": "2000-01-01_00-00-10636562792",
 "xs1": "",
 "dns": "",
 "ssid": "SSIDTEST",
 "cpheaders": "",
 "cpurl": "",
 "key": "SSIDPASSWORD",
 "spi": 0,
 "subnet": "",
 "xi1": 0,
 "xi2": 0,
 "cpdata": "",
 "mode": 0,
 "xi3": 0,
 "xs3": "",
 "gateway": "",
 "rst": "",
 "cpmeth": "POST",
 "fbkey": ""
}
```


8.5.16 PROTOCOL

Methods: [GET]

URI: <base url>/protocol

Body: GET: No body or headers allowed

Actions: List available AurAir Base protocol adapters data.

Firmware version: 1.04

JSON data example:


```
[
 {
  "xs1": "",
  "template": "domoticz_mqtt.html",
  "xi3": 0,
  "xi2": 0,
  "id": 2,
  "module": "domoticz_mqtt",
  "name": "Domoticz MQTT",
  "timestamp": "2019-08-18_13-53-27859068510",
  "xs2": "",
  "xs3": "",
  "xi1": 0,
  "protocol": "MQTT"
 }
]
```


8.5.17 CONTROLLER

Methods: [GET, POST, PUT]

URI: <base url>/controller

Body: GET: No body or headers allowed

PUT: JSON Body containing only changed data (and the timestamp!)

POST: JSON Body containing new data (without the timestamp!)

Actions: List, update or insert AurAir Base controllers configuration data. A reboot is needed to make a change effective.

Firmware version: 1.04

WARNING: CHANGING THE AZURE CONTROLLER DATA WILL MAKE IT FAIL

JSON data example:


```
[
 {
  "hostname": "192.168.72.15",
  "publish": "",
  "protocol": "Domoticz HTTP",
  "user": "",
  "id": 2,
  "enable": "on",
  "usedns": "",
  "port": 8060,
  "subscribe": "",
  "password": "",
  "timestamp": "2019-08-18_15-20-1748655722"
 }
]
```


8.5.18 PLUGIN

Methods: [GET]

URI: <base url>/plugin

Body: GET: No body or headers allowed

Actions: List AurAir Base plugin configuration data.

Firmware version: 1.04

JSON data example:


```
[
 {
  "xs2": "",
```



```
  "xs3": "",
  "valuecnt": 1,
  "stype": "",
  "port": "",
  "timestamp": "2000-01-01_00-00-13792662289",
  "module": "displaybtn",
  "name": "Display Button",
  "senddata": "",
  "formula": "",
  "xi3": 0,
  "xi1": 0,
  "xi2": 0,
  "template": "displaybtn.html",
  "inverse": "",
  "id": "5",
  "dtype": "",
  "pullup": "",
  "xs1": "",
  "sync": "",
  "timer": "",
  "pincnt": 1
 }
]
```


8.5.19 DEVICE

Methods: [GET]

URI: <base url>/device

Body: GET: No body or headers allowed

Actions: List AurAir Base sensor device configuration data.

Firmware version: 1.04

WARNING: DO NOT CHANGE DEVICE CONFIGURATION DATA!

JSON data example:


```
[
 {
  "bootstate": "off",
  "xs2": "",
  "dxpin": "d35",
  "valuedecimal": "",
  "pullup": "off",
  "id": 5,
  "valuesubscription": "",
  "timestamp": "2000-01-01_00-00-24225957641",
  "xi1": 0,
  "xi2": 0,
  "name": "dbtn",
  "xi3": 0,
  "controller": 0,
  "controllerid": 0,
  "xs3": "",
  "inverse": "off",
  "uart": 0,
  "pluginid": "5",
  "delay": 60,
  "valueformula": "",
  "valuename": "",
  "sync": "off",
  "xs1": "",
  "i2c": 0,
  "enable": "on",
  "spi": 0,
```

"port": ""
}
]

8.5.20 CO2

Methods: [GET]

URI: <base url>/device/co2

Body: GET: No body or headers allowed

Actions: List the latest CO2 value the AurAir Base sensor has measured.

Firmware version: 1.04

JSON data example:

{
"co2": 691
}

8.5.21 HUMIDITY

Methods: [GET]

URI: <base url>/device/humidity

Body: GET: No body or headers allowed

Actions: List the latest Humidity value the AurAir Base sensor has measured.

Firmware version: 1.05

JSON data example:

{
"humidity": "50.3"
}

8.5.22 CALIBRATE CO2 SENSOR

Methods: [GET]

URI: <base url>/device/co2/calibrate

Body: GET: No body or headers allowed

Actions: Calibrate the AurAir Base co2 sensor.

Firmware version: 1.05

JSON data example:

None