
Control Sonoff Devices from Home Assistant

HACS Default

Home Assistant custom component for control Sonoff devices with eWeLink (original) firmware over LAN and/or Cloud.

New features in version 3.0

- support Integration UI, Devices and Zones
- support new eWeLink API
- support multiple eWeLink accounts and homes
- support many sensors for each device (include RFBridge)
- support thermostats for Sonoff TH ans NS Panel
- support preventing DB size growth
- support many new Hass features

Features from previous versions

- can manage **both local and cloud control at the same time!**
- support old devices with 2.7 firmware (only cloud connection)
- support new device types: color lights, sensors, covers
- support eWeLink cameras with PTZ
- support unavailable device state for both local and cloud connection
- support sensors for Sonoff RF Bridge 433
- support ZigBee Bridge and Devices
- added new debug mode for troubleshooting

Pros

- work with original eWeLink / Sonoff firmware, no need to flash devices
- work over Local Network and/or Cloud Server
- work with devices without DIY-mode
- work with devices in DIY-mode
- support single and multi-channel devices
- support TH and Pow device sensors
- support Sonoff RF Bridge 433 for receive and send commands
- support Sonoff GK-200MP2-B Camera
- instant device state update with local Multicast or cloud Websocket connection
- load devices list from eWeLink Servers (with names and encryption keys) and save it locally
- (optional) change device type from **switch** to **light**

Component review from DrZzs



There is another great component by @peterbuga, that works with cloud servers.

Thanks to @beveradb and @mattsaxon for researching the local Sonoff protocol. Thanks to @michthom and @EpicLPer for researching the local Sonoff Camera protocol.

Tested Devices

Almost any single or multi-channel Switch working in the eWeLink application will work with this Integration even if it is not on the list.

Tested (LAN and Cloud)

These devices work both on a local network and through the cloud.

- Sonoff Basic, BASICR2, BASICR3, RFR2, RFR3
- Sonoff Mini/MINIR2, MINI R3 (no need use DIY-mode)
- Sonoff Micro
- Sonoff TH10/TH16 (support Thermostat)
- Sonoff 4CH, 4CHR2, 4CHR3 & 4CHPROR3
- Sonoff POWR2 (show power consumption)
- Sonoff DUALR3/DUALR3 Lite
- Sonoff RF Bridge 433 (receive and send commands) fw 3.5.0
- Sonoff D1 (dimmer with brightness control) fw 3.4.0, 3.5.0
- Sonoff G1 fw 3.5.0
- Sonoff Dual
- Sonoff iFan02, iFan03, iFan04 (light and fan with speed control) fw 3.4.0
- Sonoff S20, S26, S31, S40 fw 1.3, 1.4, S55

-
- Sonoff SV fw 3.0.1
 - Sonoff T1, TX Series
 - Sonoff T4EU1C
 - Sonoff IW100/IW101
 - Sonoff Slampher R2
 - Sonoff 5V DIY
 - Sonoff RE5V1C
 - Sonoff NSPanel
 - MiniTiger Wall Switch (I have 8 without zero-line) fw 3.3.0
 - Smart Circuit Breaker, [link](#), [link](#)
 - Smart Timer Switch
 - Eachen WiFi Smart Touch fw 3.3.0

Tested (only Cloud)

These devices only work through the cloud!

- Sonoff POW (first) fw 2.6.1
- Sonoff L1 (color, brightness, effects) fw 2.7.0
- Sonoff B1 (color, brightness, color temp) fw 2.6.0
- Sonoff B02, B05-B, B05-BL
- Sonoff SC (five sensors) fw 2.7.0
- Sonoff DW2
- Sonoff SwitchMan R5
- Sonoff S-MATE
- Sonoff S40 fw 1.1
- King Art - King Q4 Cover (pause, position) fw 2.7.0
- KING-M4 (brightness) fw 2.7.0
- Eachen WiFi Door/Window Sensor
- Essential Oils Diffuser (fan and color light) fw 2.9.0
- Smart USB Mosquito Killer
- Smart Bulb RGB+CCT

Tested ZigBee (only Cloud)

- Sonoff ZigBee Bridge - turn on for pairing mode
- SONOFF SNZB-01 - Zigbee Wireless Switch
- SONOFF SNZB-02 - ZigBee Temperature and Humidity Sensor
- SONOFF SNZB-03 - ZigBee Motion Sensor
- SONOFF SNZB-04 - ZigBee Wireless door/window sensor

Tested Cameras (only LAN)

Maybe other eWeLink cameras also work, I don't know.

- Camera GK-100CD10B (camera with PTZ)
- Sonoff GK-200MP2-B (camera with PTZ)

Installation

HACS > Integrations > Plus > **SonoffLAN**

Or manually copy `sonoff` folder from latest release to `custom_components` folder in your config folder.

Configuration

Configuration > Integrations > Add Integration > Sonoff

If the integration is not in the list, you need to clear the browser cache.

You can setup multiple integrations with different ewelink accounts.

Important. If you use the same account in different smart home systems, you will be constantly unlogged from everywhere. In this case, you need to create a second ewelink account and share your devices or home with it.

- Problems: another Home Assistant, Homebridge, eWeLink addon, etc.
- No Problems: latest eWeLink mobile app v4+

Issues

Before posting new issue:

1. Check the number of online devices on the System Health page
2. Check warning and errors on the Logs page
3. Check **debug logs** on the Debug page (must be enabled in integration options)
4. Check **open and closed** issues
5. Share integration diagnostics (supported from Hass v2022.2):
 - All devices: Configuration > Integrations > **Sonoff** > 3 dots > Download diagnostics
 - One device: Configuration > Devices > Device > Download diagnostics

There is no private data, but you can delete anything you think is private.

Configuration UI

Configuration > Integrations > **Sonoff** > Configure

Mode

In **auto** mode component using both local and cloud connections to your devcies. If device could be reached via LAN - the local connection will be used. Otherwise the cloud connection will be used.

local mode or **cloud** mode will use only this type of connection.

Sometimes it can be difficult to get a local connection to work. You need a local network with working Multicast (mDNS/zeroconf) traffic between the Hass and your devices. Read about common problems.

Each time the integration starts, a list of user devices is loaded from cloud and saved locally (`/config/.storage/sonoff/`).

auto mode and **local** mode can work without Internet connection. If the integration fails to connect to the cloud - the component will use the previously saved list of devices and continue to work only in **local** mode. **auto** mode will continue trying to connect to the cloud.

local mode can't work without ewelink credentials because it needs devices encryption keys.

Devices in DIY mode can be used without ewelink credentials because their protocol unencrypted.

It is **highly recommended** that you use `mode: auto` and do not use `mode: local` or DIY mode. Because the local protocol is not always stable and you will get a bad experience. Devices may sometimes disappear from the network or fail to respond to local requests. Also some POW and TH devices cannot update their sensors without a cloud connection.

Debug page

Enable debug page in integration options. Reload integrations page. Open: Integraion > Menu > Known issues.

Debug page shows only integration logs and removes some private data. You can filter log and enable auto refresh (in seconds).

```
1 http://192.168.1.123:8123/api/sonoff/c8503fee-88fb-4a18-84d9-abb782bf0aa7?q=1000xxxxxx&r=2
```

Homes

By default component loads cloud devices **only for current active Home** in ewelink application. If there is only one Home in the account, it shouldn't be a problem. Otherwise you can select one or multiple Homes to load devices from.

Configuration YAML

These settings are made via YAML.

Important. DeviceID is always 10 symbols string from entity_id or eWeLink app.

Custom device_class

You can convert all switches into light by default:

```
1 sonoff:
2   default_class: light # (optional), default switch
```

You can convert specific switches into `light`, `fan` or `binary_sensor`:

```
1 sonoff:
2   devices:
3     1000xxxxxx:
4       device_class: light
5       name: Sonoff Basic
6     1000yyyyyy:
7       device_class: fan
8       name: Sonoff Mini
```

You can convert multi-channel devices (e.g. Sonoff T1 2C):

```
1 sonoff:
2   devices:
3     1000xxxxxx:
4       device_class: [light, fan]
5       name: Sonoff T1 2C
6     1000yyyyyy:
7       device_class: [switch, light]
8       name: MiniTiger 2CH
```

You can convert multi-channel device (e.g. Sonoff T1 3C) into single light with brightness control:

```
1 sonoff:
2   devices:
3     1000xxxxxx:
```

```
4     device_class:
5       - light: [1, 2, 3]
6     name: Sonoff T1 3C
```

You can control multiple light zones with single multi-channel device (e.g. Sonoff 4CH):

```
1  sonoff:
2    devices:
3      1000xxxxxx:
4        device_class:
5          - switch: 1 # entity 1 (channel 1)
6          - light: [2, 3] # entity 2 (channels 2 and 3)
7          - fan: 4 # entity 3 (channel 4)
8        name: Sonoff 4CH
```

You can change `device_class` for Binary Sensor:

```
1  sonoff:
2    devices:
3      1000xxxxxx:
4        device_class: window
```

You can change `device_class` for Cover:

```
1  sonoff:
2    devices:
3      1000xxxxxx:
4        device_class: shutter
```

You can set the `uuid` when running in DIY mode to enable the device features. More info [here](#).

```
1  sonoff:
2    devices:
3      1000xxxxxx:
4        extra: { uuid: 136 } # Sonoff B05-BL
```

Custom devices

```
1  sonoff:
2    devices:
3      1000xxxxxx:
4        name: Device name from YAML # optional rewrite device name
5        host: 192.168.1.123 # optional force device IP-address
6        devicekey: xxx # optional encryption key (downloaded
                        automatically from the cloud)
```

Custom sensors

If you want some additional device attributes as sensors:

```
1 sonoff:
2   sensors: [staMac, bssid, host]
```

Force update

You can request actual device state and all its sensors manually at any time using `homeassistant.update_entity` service. Use it with any device entity except sensors. Use it with only one entity from each device.

As example, you can create an automation for forced temperature updates for Sonoff TH:

```
1 trigger:
2   - platform: time_pattern
3     minutes: '3'
4 action:
5   - service: homeassistant.update_entity
6     target:
7       entity_id: switch.sonoff_1000xxxxxx
```

Preventing DB size growth

Pow devices may send a lot of data every second. You can reduce the amount of processed data.

For multi-channel devices use `power_1`, `current_2`, etc.

```
1 sonoff:
2   devices:
3     1000xxxxxx:
4       reporting:
5         power: [30, 3600, 1] # min seconds, max seconds, min delta
6           value
7         current: [5, 3600, 0.1]
8         voltage: [60, 3600, 5]
```

- if new value came before `min seconds` - it will be “delayed”
- if new value came between `min` and `max seconds`
 - if delta lower than `delta value` - it will be “delayed”
 - otherwise - it will be used
- if new value came after `max seconds` - it will be used

-
- any used value will erase “delayed” value
 - new “delayed” value will overwrite old one
 - “delayed” value will be checked for the above conditions every 30 seconds

Sonoff Pow

Support **power**, **current** and **voltage** sensors via LAN and Cloud connections. Also support energy (consumption) sensor only with **Cloud** connection.

Many models of Sonoff power devices DON'T send **power**, **current** and **voltage** by default. You need to ASK these devices to send this data. This can ONLY be done through a cloud-based request. The mobile app does it. And the integration does it (only in the **auto** and **cloud** modes).

By default **energy** data loads from cloud every hour. You can change interval via YAML and add history data to sensor attributes (max size - 30 days, disable - 0). For multi-channel devices use **energy_1**, **energy_2**.

```
1 sonoff:
2   devices:
3     1000xxxxxx:
4       reporting:
5         energy: [3600, 10] # update interval (seconds), history size (
6                               days)
7   template:
8     - sensor:
9       - name: "10 days consumption"
10         unit_of_measurement: "kWh"
11         state: "{{ (state_attr('sensor.sonoff_1000xxxxxx_energy', '
12                               history') or [])|sum }}"
```

You can also setup a integration sensor, that will collect energy data locally by Hass:

```
1 sensor:
2   - platform: integration
3     source: sensor.sonoff_1000xxxxxx_power
4     name: energy_spent
5     unit_prefix: k
6     round: 2
```

Sonoff TH

Support optional Climate entity that controls Thermostat. You can control low and high temperature values and hvac modes:

-
- **heat** - lower temp enable switch, higher temp disable switch
 - **cool** - lower temp disable switch, higher temp enable switch
 - **dry** - change control by **humidity** with previous low/high switch settings

In **dry** mode, the Thermostat controls and displays Humidity. But the units are displayed as temperature (Hass limitation).

Thermostat can be controlled only with **Cloud** connection. Main switch and TH sensors support LAN and Cloud connections.

Sonoff RF Bridge 433

RF Bridge support learning up to 64 signals (16 x 4 buttons).

Video HOWTO from @KPeyanski



Important. Integration v3 supports automatic creation of sensors for RF Bridge. All **buttons** will be created as Button entity. All **alarms** will be created as Binary sensor.

Both button and binary sensor has `last_triggered` attribute with the time of the last signal received. You can use it in automations.

Binary sensor will stay in **on** state during **120 seconds** by default. Each new signal will reset the timer. Binary sensor support restore state between Hass restarts.

If you has door sensor with two states (for open and for closed state) like this one, you can config `payload_off` as in the example below. Also disable the timeout if you do not need it in this case (with `timeout: 0` option).

You can use any `device_class` that is supported in Binary Sensor. With `device_class: button` you can convert sensor to button.

PIR Sensor

```
1 sonoff:
2   rfbridge:
3     PIR Sensor 1: # button/alarm name in eWeLink application
4     device_class: motion
5     timeout: 60 # optional (default 120), timeout in seconds for
                  auto turn off
```

Single State Sensor

```
1 sonoff:
2   rfbridge:
3     Door Sensor 1: # button/alarm name in eWeLink application
4     name: Door Sensor # optional, you can change sensor name
5     device_class: door # e.g. door, window
6     timeout: 5
```

Dual State Sensor

```
1 sonoff:
2   rfbridge:
3     Sensor1: # button/alarm name in eWeLink application (open signal)
4     name: Window Sensor # optional, you can change sensor name
5     device_class: window # e.g. door, window
6     timeout: 0 # disable auto close timeout
7     payload_off: Sensor2 # button/alarm name in eWeLink application
                      (close signal)
```

You can read more about using this bridge in [wiki](#).

Sonoff GK-200MP2-B Camera

Currently only PTZ commands are supported. Camera entity is not created now.

You can send `left`, `right`, `up`, `down` commands with `sonoff.send_command` service:

```
1 script:
2   left:
3     sequence:
4     - service: sonoff.send_command
5       data:
6         device: '012345' # use quotes, this is important
7         cmd: left
```

`device` - this is the number from the camera ID `EWLK-012345-XXXXX`, exactly 6 digits (leading zeros - it is important).

Common problems in only LAN mode

`auto` mode and `cloud` mode users don't have these problems.

Devices are not displayed

- not all devices supports local protocol
- two routers
- **docker** with port forwarding
 - you must use: `--network host`
 - hassio users are okay
- **virtual machine** with port forwarding
 - you must use bridge virtual network mode (not NAT mode)
- Oracle VM VirtualBox
- linux firewall
- linux network driver
- incorrect network interface selected in Configuration > Settings > Global > Network

The devices publish their data through Multicast DNS (mDNS/zeroconf), read more.

Devices unavailable after reboot

All devices **unavailable** after each Home Assistant restart. Devices are automatically detected in the local network after each restart. Sometimes devices appear quickly. Sometimes after a few minutes. If this does not happen, there are some problems with the multicast / router.

Raw commands

The component adds the service `sonoff.send_command` to send low-level commands.

Example service params to single switch:

```
1 device: 1000xxxxxx
2 switch: 'on'
```

Example service params to multi-channel switch:

```
1 device: 1000xxxxxx
2 switches: [{outlet: 0, switch: 'off'}]
```

Example service params to dimmer:

```
1 device: 1000123456
2 cmd: dimmable
3 switch: 'on'
4 brightness: 50
5 mode: 0
```

Getting devicekey manually

The average user does not need to get the device key manually. The component does everything automatically, using the ewelink account.

1. Put the device in setup mode
2. Connect to the Wi-Fi network **ITEAD-10000**, password12345678
3. Open in browser <http://10.10.7.1/device>
4. Copy **deviceid** and **apikey** (this is **devicekey**)
5. Connect to your Wi-Fi network and setup Sonoff via the eWeLink app

Useful Links

- <https://github.com/peterbuga/HASS-sonoff-ewelink>
- <https://github.com/beveradb/sonoff-lan-mode-homeassistant>
- <https://github.com/mattsaxon/sonoff-lan-mode-homeassistant>
- https://github.com/EpicLPer/Sonoff_GK-200MP2-B_Dump
- <https://github.com/bwp91/homebridge-ewelink>
- <https://blog.ipsumdomus.com/sonoff-switch-complete-hack-without-firmware-upgrade-1b2d6632c01>
- https://github.com/itead/Sonoff_Devices_DIY_Tools
- SONOFF DIY MODE API PROTOCOL
- No Tasmota And EWeLink Cloud To Control The SONOFF Device? YES!