SONOFF DIY MODE API PROTOCOL

- Version: 1.4.0
- Date: 2019-07-31

Introduction

This API protocol is appropriate only for the DIY MODE device (Basic R3/RFR3/Mini) with firmware version of 3.3.0 or above

Working Flow:

Set up a specified SSID and password WiFi network, enter the DIY MODE by switching the GPIO 16 to Low-level (jumper plugged-in) , discover the device IP address through mDNS service, control and operate the functions of the device according to the API that we offer.

New device is in eWeLink mode as default, if you want to use this API protocol to control the device, the device must be switched into DIY MODE.

Mode Switch	Operation Mode
Without Jumper (factory default)	eWeLink MODE
With Plugged-in Jumper	DIY MODE

DIY MODE Description

How to enter the DIY MODE:

1. Unscrew the screws on the bottom, open the bottom lid;
2. Plug-in the jumper on GPIO 16.

PS: you must disconnect the device from the power supply before you operate DIY MODE switch.

DIY MODE LED blinking:

1. Fast single blinking -- The device does not connect to the WiFi (Router or Hotspot);
2. Fast double blinking -- The device connects to the WiFi successfully able to be discovered through mDNS and respond the request from LAN network.

DIY MODE functions:

In DIY MODE, Any application can send API to control the device via LAN. (refer to API instruction page)

Note：

1. The user settings will be cleaned when change the operation mode from one to another. e.g. the device was in DIY MODE with a specific SSID and PW info setting, users would like to

make a factory reset for the device SSID and PW, the steps would be, switch into eWeLink Mode, power on the device, at this time, the user setting of the device will be cleaned.
2. To switch into eWeLink Mode, you need to disconnect the device from the power supply first, plug-out the jumper for GPIO 16 high level.

Set up the DIY MODE operating environment

1. LAN (WiFi) requirement and setting

Requirement: 2.4GHz wifi and support mDNS service

Setting：

WiFi SSID: `sonoffDiy`

WiFi password: `20170618sn`

2. Device enter DIY MODE

First step: Disconnect the device from the power supply, unscrew the screws on the bottom, open the bottom lid, Plug-in the jumper on GPIO 16.

Second Step: Assembly the enclosure of the product, power on the device, it will automatically connect to the specific WiFi (SSID: `sonoffDiy` and password: `20170618sn`)

LED blinking cases:

- Fast double blinking indicates that the device enter the DIY MODE and connect to the WiFi (SSID: `sonoffDiy` and password: `20170618sn`) successfully.
- Fast single blinking indicates that the device not connecting to WiFi. please check the WiFi SSID and password was set successfully or not, please check whether your firmware version is 3.3.0 (or above), check the WiFi router or hotspot support mDNS service or not.

3. Test environment set up

3.1. Be sure that your PC and DIY MODE device connect to the same router (hotspot) and LAN

3.2. Open DIY_TOOL.exe

If the device was discovered in DIY_TOOL.exe, it means that **mDNS service is working** (you are able to discover the DIY MODE device via LAN), go ahead to the next step.

If the device was not discovered in DIY_TOOL.exe，please check：

- PC and DIY MODE device are in the same router and LAN or not
- Device is in DIY MODE and fast double blinking or not, if the device is not in DIY MODE, go to previous step of Device enter DIY MODE
- The router (or hotspot) supports mDNS or not
- If you PC has dual NIC, please set the NIC which is in the same LAN with the DIY MODE device as the first one.

3.3. Test ON/OFF function

1. click “ON”----------the device will be ON.
2. click “OFF”----------the device will be OFF.

3.4. Test Power On State function

1. click “Power-up-state-ON”-------- the device is ON when power supply is recovered.
2. click “Power-up-state-OFF”-------- the device is OFF when power supply is recovered.
3. click “Power-up-state-KEEP”--------the device status keeps as the same as the state before power supply is gone.

3.5. Test WiFi signal strength function

1. choose the device in the device list, click "signal", WiFi signal strength is shown.
2. get the device WiFi signal strength from mDNS text record. (mDNS txt record refer to 4.1)

3.6. Test Inching function

1. click"Inching"
2. switch the inching function on
3. input minutes
4. input second
5. with a extra 0.5 second or not
6. click "OK"

3.7. Test WiFi SSID and password setting

1. choose the device in the device list
2. click "change SSID password"
3. input a SSID you want to connect
4. input a corresponding password
5. click "OK"

3.8. Test Get Device Info function

1. choose the device in the device list
2. click "info"

4. DIY MODE LAN discovery mechanism

DIY MODE LAN discovery implements IETF Multicast DNS protocol and DNS-Based Service Discovery protocol. [1]

4.1. Device mDNS service info publish process

The device publishes its own service (i.e. device capability) according to the mDNS/DNS-SD standard discovery protocol when the device is connected to LAN (Local Area Network).

The fields defined by eWeLink are as follows:

Attribute	Description	Example
IP Address	The LAN IP Address is obtained through DHCP instead of the Link-Local address of IPv4/IPv6	
Hostname	The Hostname must be unique in LAN; Format: eWeLink_[Device ID]	eWeLink_10000000d0
Service Type	ewelink._tcp	
Service Instance Name	The Service Instance Name must be unique in LAN; Max: 63 bytes (21 UTF8 Characters)	
TXT Record	One or more strings; No exceeded 255 bytes for each string; No exceeded 1300 bytes for the entire TXT record;	

- **TXT Record note**：

    1. TXT Record must contain below strings:

    "txtvers=1", "id=[device ID]", "type=[device type]", "apivers=[device API interface version]", "seq=[TXT Record serial number]", "data1=[device information]";

    2. Optional strings:

    "data2=[device information]", "data3=[device information]", "data4=[device information]"

    3. “seq=[TXT record sequence number]” indicates the order in which the TXT records are updated (the order in which the device status is updated). It is recommended to be a positive integer that increments from 1 (reset to 1 when the device restarts);
    4. When the device information is longer than 249 bytes, the first 249 bytes must be stored in data1, and the remaining bytes are divided by length 249, which are stored in data2, data3, and data4. The complete device information format is a JSON object, for instance:

    data1=

    {"switch":"on","startup":"stay","pulse":"on","pulseWidth":2000,"ssid":"eWeLink","otaUnlock":true}

Whenever content other than seq changes, such as Service Instance Name is modified, device information is updated, etc., the device must multicast the corresponding DNS record (including the incremented seq) according to the mDNS/DNS-SD standard.

4.2. Discovery Process for Device Service

The discovery process must follow the mDNS/DNS-SD Discovery protocol to discover the Sonoff DIY MODE device with “*ewelink*.tcp” service type when your application or client connect with Internet (WiFi or Ethernet);

Here is the discovery process:

1. Search in the LAN for all devices with the service type *ewelink*.tcp through the DNS PTR record.
2. Get the Hostname and Port of device service via parsing out the device DNS SRV record.
3. Get the info of “device ID”, “Service Type”, “device API interface version” and “device information” via parsing out the device DNS TXT Record.

Note:

1. When the “device type” of the device service does not match with the “device type” of your application or client, or the device API interface version of the device service is higher than your application or client’s, the application or client should not parse out the “device information” and call the device API interface, but prompt the specific reason for users why the device cannot be controlled via LAN and suggest to upgrade the application or client.
2. The application or client get the IP address of the device via DNS A record when the device API interface is about to be called.

4.3. Device info by runing `mdns.py` file

(Discover the device via mDNS and get the info of device)

```
1000806ace  192.168.1.104  8081  {b'txtvers': b'1', b'id': b'1000806ace',
b'type': b'diy_plug', b'apivers': b'1', b'seq': b'853',
b'data1':b'{"switch":"off","startup":"off","pulse":"off","pulseWidth":500,"rssi"
:-70}'}
```

Device API Control Protocol (HTTP POST)

The device must open the HTTP server in the port declared by the DNS SRV record before the device publishes its services; the device publishes the capabilities through a HTTP-based RESTful API. Because of the LAN's security and device's limited computing power, this document recommends that the device provides HTTP instead of HTTPS interface.

The device type is diy_plug(type=diy_plug) and the device API interface version is 1 (apivers=1).

RESTful API Request and Response Format

URL: http://[ip]:[port]/[path]

Return value format: json

Method: HTTP post

Attribute	Type	Example	Optional	Description
deviceid	String	100000140e	No	The device ID for this request. After receiving the request, the device must determine whether the deviceid is supported or not
data	Object	{"switch": "on"}	No	Object type, Specific device information setting when controlling the device, null when check the device information

RESTful API Request works in POST method and JSON formatted request body.

```
{
    "deviceid": "100000140e",
    "data": {
       "switch": "on"
       }
}
```

Attribute	Type	Optional	Description
deviceid	String	No	The device ID for this request. After receiving the request, the device must determine whether the deviceid is supported or not
data	Object	No	Object type, Specific device information setting when controlling the device, null when check the device information

RESTful API Response works in 200 OK HTTP response code and JSON formatted response body.

```
{
   "seq": 2,
   "error": 0,
    "data": {
        "signalStrength": -67
    }
}
```

Attribute	Type	Optional	Description
seq	Number	No	The order of device status update (also the order of TXT Record update)
error	Number	No	Whether the device successfully sets the specified device information. - **0:** successfully - **400:** The operation failed and the request was formatted incorrectly. The request body is not a valid JSON format. - **401:** The operation failed and the request was unauthorized. Device information encryption is enabled on the device, but the request is not encrypted. - **404:** The operation failed and the device does not exist. The device does not support the requested deviceid. - **422:** The operation failed and the request parameters are invalid. For example, the device does not support setting specific device information.
data	Object	No	Object type, it returns specific device info when check the device information

Note: Due to the device computing capability, the time interval of each HTTP request should be no less than 200ms.

1. ON/OFF status

URL: http://[ip]:[port]/zeroconf/switch

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": {
        "switch": "on"
    }
 }
```

Attribute	Type	Optional	Description
switch	String	No	**on:** turn the switch on, **off:** turn the switch off

2. Power On State

URL: http://[ip]:[port]/zeroconf/startup

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": {
        "startup": "stay"
    }
 }
```

Attribute	Type	Optional	Description
startup	String	No	**on:** the device is on when power supply is recovered. **off:** the device is off when power supply is recovered. **stay:** the device status keeps as the same as the state before power supply is gone

3. WiFi Signal Strength

URL: http://[ip]:[port]/zeroconf/signal_strength

Return value format: json

Method: HTTP post

Request body

e.g.

```
{
    "deviceid": "100000140e",
    "data": { }
 }
```

Null, no string is required

Response body

e.g.

```
{
    "seq": 2,
    "error": 0,
    "data": {
        "signalStrength": -67
    }
}
```

Attribute	Type	Optional	Description
signalStrength	Number	No	The WiFi signal strength currently received by the device, negative integer, dBm

4. Inching

URL: http://[ip]:[port]/zeroconf/pulse

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": {
        "pulse": "on",
        "pulseWidth": 2000
    }
 }
```

Attribute	Type	Optional	Description
pulse	String	No	**on:** activate the inching function; **off:** disable the inching function
pulseWidth	Number	Yes	Required when "pulse" is on, pulse time length, positive integer, ms, only supports multiples of 500 in range of 500~36000000

5. WiFi SSID and Password Setting

URL: http://[ip]:[port]/zeroconf/wifi

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": {
        "ssid": "eWeLink",
        "password": "WeLoveIoT"
    }
 }
```

Attribute	Type	Optional	Description
ssid	String	No	SSID of the WiFi network to which the device will connect
password	String	No	password of the WiFi network to which the device will connect

6. OTA Function Unlocking

URL: http://[ip]:[port]/zeroconf/ota_unlock

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": { }
 }
```

Note: Null, no string is required

The following failure codes are added to the error field of the response body:

- **500:** The operation failed and the device has errors. For example, the device ID or API Key error which is not authenticated by the vendor's OTA unlock service;

- **503:** The operation failed and the device is not able to request the vendor's OTA unlock service. For example, the device is not connected to WiFi, the device is not connected to the Internet, the manufacturer's OTA unlock service is down, etc.

7. OTA New Firmware

URL: http://[ip]:[port]/zeroconf/ota_flash

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": {
        "downloadUrl": "http://192.168.1.184/ota/new_rom.bin",
        "sha256sum":
"3213b2c34cecbb3bb817030c7f025396b658634c0cf9c4435fc0b52ec9644667"
    }
 }
```

Attribute	Type	Optional	Description
downloadUrl	String	No	The download address of the new firmware, only supports the HTTP protocol, the HTTP server must support the Range request header.
sha256sum	String	No	SHA256 checksum (hash) of the new firmware, it is used to verify the integrity of the new firmware downloaded

The following failure codes are added to the error field of the response body:

- **403:** The operation failed and the OTA function was not unlocked. The interface "3.2.6OTA function unlocking" must be successfully called first.

- **408:** The operation failed and the pre-download firmware timed out. You can try to call this interface again after optimizing the network environment or increasing the network speed.

- **413:** The operation failed and the request body size is too large. The size of the new OTA firmware exceeds the firmware size limit allowed by the device.

- **424:** The operation failed and the firmware could not be downloaded. The URL address is unreachable (IP address is unreachable, HTTP protocol is unreachable, firmware does not exist, server does not support Range request header, etc.)

- **471:** The operation failed and the firmware integrity check failed. The SHA256 checksum of the downloaded new firmware does not match the value of the request body's sha256sum field. Restarting the device will cause bricking issue.

Note:

The maximum firmware size is 508KB.

The SPI flash read mode must be DOUT

8. Get Device Info

URL: http://[ip]:[port]/zeroconf/info

Return value format: json

Method: HTTP post

e.g.

```
{
    "deviceid": "100000140e",
    "data": { }
 }
```

Response body

```
{
    "seq": 2,
    "error": 0,
    "data": {
    "switch": "on",
    "startup": "stay",
    "pulse": "on",
    "pulseWidth": 2000,
    "ssid": "eWeLink",
    "otaUnlock": true
    }
 }
```

Note:

Null, no string is required

Monitor and parse the device's DNS TXT record to get the device information in real time.

Reference:

1. Multicast DNS protocol: IETF RFC 6762, https://tools.ietf.org/html/rfc6762
2. DNS-Based Service Discovery protocol: IETF RFC 6763, https://tools.ietf.org/html/rfc6763
3. Zero Configuration Networking: Zeroconf, http://www.zeroconf.org/
4. Apple Bonjour Network Discovery and Connectivity: https://developer.apple.com/bonjour/
5. Android Network Service Discovery: https://developer.android.com/training/connect-devices-wirelessly/nsd
6. Sonoff DIY Mode Demo Application on Github: https://github.com/itead/Sonoff_Devices_DIY_Tools

7. Wikipedia Zero Configuration Networking: https://en.wikipedia.org/wiki/Zero-configuration_networking
8. How does Zeroconf compare with Viiv/DLNA/DHWG/UPnP: http://www.zeroconf.org/ZeroconfAndUPnP.html