SONOFF DIY MODE API PROTOCOL

- Version: 2.0
- Date: 2020-03-23

SONOFF DIY Mode Overview

Do you want to control the device by your own app or web? DIY mode helps!

The DIY Mode is designed for IoT home automation users and developers who would like to control the SONOFF device via existing home automation open-source platform or local HTTP client instead of eWeLink App. In DIY Mode, when the device is connected with the network, it will publish its services and capabilities according to the mDNS/DNS-SD standard. Before publishing the service, the device has enabled the HTTP server on the port declared by the DNS SRV record. Device exposes the capabilities through an HTTP-based RESTful API. Users can obtain device information, control the device by sending an HTTP API request.

Supported Device

Product Class	Device	Firmware	Note
Single Channel DIY Plug	Basic R3 RF R3 MINI	3.5.0	Please refer to protocol V2.0
Single Channel DIY Plug	Basic R3 RF R3 MINI	3.3.0 - 3.4.0	Please refer to protocol V1.4

eWeLink Mode and DIY Mode

The SONOFF devices*can work in either eWeLink mode or DIY mode, In eWeLink mode, the device is connected with eWeLink cloud and controlled by eWeLink APP, while in DIY mode, device publishes its capability service and is controlled by HTTP Post request.

The steps of entering the DIY mode and connecting to an existing WiFi network:

1. Entering the Compatible Pairing Mode (AP) by long press the paring button for 5 seconds after power on
2. Connecting the Access Point named ITEAD-XXXXXXXX with default password 12345678 via mobile phone or PC
3. Browser visits http://10.10.7.1/
4. Filling in the existing WiFi network SSID and password
5. Entering DIY mode successfully with specific WiFi network connected.

Example for Single Channel DIY Plug (BASIC R3, RF R3, MINI) enters DIY mode:

1. Power on;
2. Long press the button for 5 seconds for entering Compatible Pairing Mode (AP)

   User tips: If the device has been paired with eWeLink APP, reset the device is necessary by long press the pairing button for 5 seconds, then press another 5 seconds for entering Compatible Pairing Mode (AP)
3. The LED indicator will blink continuously;
4. From mobile phone or PC WiFi setting, an Access Point of the device named ITEAD-XXXXXXXX will be found, connect it with default password 12345678
5. Open the browser and access http://10.10.7.1/
6. Next, Fill in WiFi SSID and password that the device would have connected with
7. Succeed, Now the device is in DIY mode.

Note:

1. The user settings will be cleaned once the operation mode is changed from one to another.
2. The WiFi router or AP should work in 2.4GHz and support mDNS service.
3. LED blinking meanings
   Fast single blinking -- The device does not connect to the WiFi network;
   Fast double blinking -- The device connects to the WiFi successfully and is able to be discovered through mDNS and respond the request from LAN network.
4. Once the device is already in DIY mode, the WiFi configuration page of http://10.10.7.1/ is not accessible.
5. If a wrong WiFi SSID or password was entered, the device will fail to connect with specific WiFi network, with 20 seconds timeout mechanism, the device stop connecting WiFi network, please try again with the example steps of 1-7.
6. Official firmware upgrade is only available in eWeLink APP.
7. SONOFF devices refer to BASICR3, RFR3, MINI.

mDNS Discovery Process

1. DIY MODE LAN discovery mechanism

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

2. 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)	same as Hostname
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.

3. 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. (The default port is 8081)
3. Get device IP address via DNS A record or by other means.
4. 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.

RESTful 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

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

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

Attribute	Type	Example	Optional	Description
deviceid	String	100000140e	Yes	The device ID for this request.
data	Object	{"switch": "on"}	No	Object type, Specific device information setting when controlling the device. Empty object 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": "",
    "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": "",
    "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": "",
    "data": { }
 }
```

Empty object, no attribute 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": "",
    "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": "",
    "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": "",
    "data": { }
 }
```

Empty object, no attribute 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": "",
    "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": "",
    "data": { }
 }
```

Empty object, no attribute is required.

Response body example

```
{
    "seq": 2,
    "error": 0,
    "data": {
        "switch": "off",
        "startup": "off",
        "pulse": "off",
        "pulseWidth": 500,
        "ssid": "eWeLink",
        "otaUnlock": false,
        "fwVersion": "3.5.0",
        "deviceid": "100000140e",
        "bssid": "ec:17:2f:3d:15:e"
    }
 }
```

Note: 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