Busy Tag
Local Network Guide

SIA BUSY TAG

Firmware Version 3.0 (Wi-Fi)
November 4, 2026

Controlling Busy Tag over the local network

Access point, station and STA+AP use cases for the built-in HTTP server

This document describes how to reach and control a Busy Tag running firmware 3.0 (Wi-Fi hardware, HW 2.0) from a phone, PC or script on the same network, without going through the Busy Tag cloud. It complements the *AT Command Manual*, which lists every command.

Contents

1 Overview **3**
1.1 When the server runs . . . 3
1.2 When a token is needed . . . 3

2 Use Case A: Direct Connection (Access Point) **4**
2.1 Network parameters . . . 4
2.2 Finding the password: the setup QR code . . . 4
2.3 Step by step . . . 4
2.4 Provisioning the device onto a router from AP mode . . . 5

3 Use Case B: Existing Network (Station) **5**
3.1 Setup over USB . . . 5
3.2 Finding the device on the LAN . . . 6
3.3 Sending commands . . . 6

4 Use Case C: Hybrid (STA+AP) **6**

5 HTTP API Reference **7**
5.1 Authentication . . . 7
5.2 Endpoints . . . 7
5.3 Response envelope . . . 7
5.4 POST /at . . . 7
5.5 GET /status . . . 8
5.6 POST /file?name=<file> . . . 8
5.7 GET /file?name=<file> . . . 8
5.8 POST /pattern . . . 8
5.9 Status codes . . . 9

6 Client Examples **10**
6.1 Python . . . 10
6.2 PowerShell . . . 10
6.3 Browser JavaScript . . . 10

7 Security Considerations **10**

8 Troubleshooting **11**

9 Command Quick Reference **12**

1 Overview

Busy Tag firmware 3.0 contains a small HTTP server that exposes the full AT command set as a REST API, plus file upload, file download and LED pattern upload. Almost anything that can be done over the USB serial port can also be done over the local network.

There are three ways the device can be placed on a local network. Which one to choose depends on who needs to reach the device and whether a router is available.

Use case	Wi-Fi mode	Server	Typical use
A. Direct (AP)	`AT+WM=2`	starts automatically	First-time setup from a phone; field use with no router
B. LAN (station)	`AT+WM=1`	manual (`AT+LS`)	PC or script control over an existing network
C. Hybrid (STA+AP)	`AT+WM=3`	starts automatically	Device online (cloud) while a phone can still connect directly

Table 1: Local network use cases

1.1 When the server runs

- **AP and STA+AP modes:** when the access point starts, the server starts on port `80` *without authentication*, because the setup app depends on it. (If the server was already running, it keeps its current settings.)
- **STA-only mode:** switching to STA-only mode stops the server. Start it again with `AT+LS=1,<port>,<token>`.
- **OFF:** the server is stopped.
- The server listens on all interfaces. In STA+AP mode it is reachable both on the AP address (`192.168.4.1`) and on the address the router assigned to the device.
- Server settings are kept in memory only. After a restart the server is configured again from the Wi-Fi mode, and settings made with `AT+LS` or `AT+LT` are lost.

1.2 When a token is needed

In firmware 3.0 authentication is a single on/off setting for the whole server, regardless of which network a request arrives on:

Server started by	Token
Access point start (AP or STA+AP mode)	Not required, on any interface. In STA+AP mode this includes requests from the router network.
`AT+LS=1,<port>,<token>`	Required on every interface: `Authorization: Bearer <token>`

Table 2: Authentication rules

STA+AP mode is open on the router network

In STA+AP mode (the factory default) the server answers requests from the router network without a token. Anyone on that network who knows the device's address can control it. Use STA+AP mode only on a network you trust, or switch to STA-only mode and enable the server with a token (use case B).

2 Use Case A: Direct Connection (Access Point)

The device creates its own Wi-Fi network. A phone or laptop joins that network and talks to the device at `192.168.4.1`. No router and no internet connection are needed. This is the mode used by the mobile app for first-time setup.

2.1 Network parameters

Parameter	Value
SSID	Device name (same as the USB name); see `AT+GDN`
Password	12 random characters generated per device on first boot / factory reset
Security	WPA2-PSK
Channel	6
Max stations	4
Device IP	`192.168.4.1` (gateway `192.168.4.1`, netmask `255.255.255.0`)
Client IPs	Assigned by the device's DHCP server, starting at `192.168.4.2`
HTTP server	`http://192.168.4.1/`, no token needed

Table 3: Default access point parameters

All of these except the IP range can be changed with:

```
AT+AP=<ssid>,<password>[,<channel>][,<max_conn>][,<auth>]
```

2.2 Finding the password: the setup QR code

When the access point starts, the display shows a QR code together with the SSID, password and IP address in plain text. The QR code encodes a setup deep link:

```
https://iot.busytag.com/s?v=1&id=<device_id>&ssid=<ssid>&pwd=<password>
    &n=<name>&fw=<fw>
```

The Busy Tag app reads the SSID and password from this link and joins the network automatically. Users without the app can type the SSID and password shown on screen into the phone's Wi-Fi settings. The base URL can be changed with `AT+SDLU=<url>`; `AT+QR=1` / `AT+QR=0` show and hide the code.

2.3 Step by step

1. Put the device into AP mode (over USB, or already done if the device is fresh):

   ```
   > AT+WM=2
   OK
   ```

2. Scan the QR code with the Busy Tag app, or join the SSID shown on the display manually.

3. Open `http://192.168.4.1/` in a browser to confirm the server is up. The page lists device name, IP, port and available endpoints.

4. Send commands (no `Authorization` header needed):

   ```
   curl -X POST http://192.168.4.1/at \
     -H "Content-Type: application/json" \
     -d '{"command": "AT+SC=127,FF0000"}'
   ```

2.4 Provisioning the device onto a router from AP mode

A common flow is to use AP mode only to hand the device the credentials of the real network, then let it switch over:

```
# 1. Optional: let the device scan, then read the results
curl -X POST http://192.168.4.1/at -d '{"command":"AT+WSCAN="}'
curl -X POST http://192.168.4.1/at -d '{"command":"AT+WSCAN?"}'

# 2. Give it the router credentials
curl -X POST http://192.168.4.1/at \
  -d '{"command":"AT+STA=MyNetwork,MyPassword"}'

# 3. Check that it got an IP (field 3)
curl -X POST http://192.168.4.1/at -d '{"command":"AT+STA?"}'

# 4. Switch to STA+AP (3) to keep the AP, or station only (1)
curl -X POST http://192.168.4.1/at -d '{"command":"AT+WM=3"}'
```

Mode changes over HTTP are applied after the response

When `AT+WM` arrives over HTTP the firmware replies `OK` first and switches the mode shortly afterwards, so the HTTP response is not lost when the access point restarts. Expect the connection to drop right after the response. In STA-only mode the server is stopped; to use the API on the router network afterwards, enable it over USB (use case B).

3 Use Case B: Existing Network (Station)

The device joins an existing router as a normal client. Any PC, server or script on the same LAN can reach it. In STA-only mode the server is off until it is started with `AT+LS`, and every request then needs the bearer token.

3.1 Setup over USB

```
> AT+STA=MyNetwork,MyPassword
OK
+evn:WIFI_CONNECTED,MyNetwork,192.168.1.42

> AT+WM=1
OK

> AT+LS=1,8080,Q7ZpX2mWc9KdT4hLr8VbN3sYf6GjA1eU
OK

> AT+LS?
+LS:1,8080,1,RUNNING
OK

> AT+GIP
+IP:192.168.1.42
OK
```

`AT+LS=<enable>,<port>,<token>`:

- `<enable>`: `1` starts the server, `0` stops it.
- `<port>`: `1024`–`65535`. Port `80` cannot be selected with `AT+LS`; only the automatic start in AP and STA+AP modes uses it.

- `<token>`: up to 63 characters, no commas. A non-empty token replaces the current one and turns authentication on. Leave it empty (`AT+LS=1,8080,`) to keep the current token and authentication setting. All three parameters are required.

`AT+LT?` returns the current token and `AT+LT` replaces it with a new random 32-character one, which takes effect for the next request. Neither is stored: after a restart, run `AT+LS` again.

Configure the server over USB

Send `AT+LS` over USB, not through the HTTP API. Applying it restarts the server while the HTTP request is still being answered, so the response is lost.

3.2 Finding the device on the LAN

The firmware does not advertise itself via mDNS/Bonjour. Use one of:

- `AT+GIP` or `AT+STA?` over USB;
- the router's DHCP client list (hostname / MAC);
- a DHCP reservation on the router so the address never changes (recommended for automation).

3.3 Sending commands

```
curl -X POST http://192.168.1.42:8080/at \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer Q7ZpX2mWc9KdT4hLr8VbN3sYf6GjA1eU" \
  -d '{"command": "AT+SP=busy.png"}'
```

4 Use Case C: Hybrid (STA+AP)

The device is connected to a router (so cloud sync and remote commands work) and at the same time keeps its own access point up. A phone can join the device's AP at any moment to reconfigure it, even if the router credentials have changed. This is the factory default mode.

- The server starts on port `80` when the access point starts.
- No token is needed, neither on the AP side (`http://192.168.4.1/`) nor on the LAN side at the address the router assigned. See the caution in Section 5.1.
- The setup QR code is shown on boot.
- The AP and the station share one radio, so the AP follows the router's channel.

5 HTTP API Reference

5.1 Authentication

When authentication is on (the server was started with a token through `AT+LS`), every endpoint except `/` and `/favicon.ico` needs:

```
Authorization: Bearer <token>
```

A missing or wrong token returns `401` with `{"status":"error","message":"Authentication required"}`. When the server was started by the access point, no endpoint needs a token. All responses carry permissive CORS headers (`Access-Control-Allow-Origin: *`) and `OPTIONS` preflight is answered on every API path, so the API can be called from a browser page.

5.2 Endpoints

Method	Path	Description
GET	`/`	HTML info page (device name, IP, port, whether a token is needed, curl examples). Always open.
GET	`/favicon.ico`	Icon. Always open.
POST	`/at`	Execute one AT command.
GET	`/status`	Server and device status as JSON.
POST	`/file?name=<file>`	Upload a file to storage (raw body).
GET	`/file?name=<file>`	Download a file from storage.
POST	`/pattern`	Replace the custom LED pattern.

Table 4: HTTP endpoints

5.3 Response envelope

`/at`, `/file` (upload) and `/pattern`, and all errors, answer with:

```
{
  "status":  "success" | "error",
  "message": "<human readable>",
  "data":    "<optional payload>"
}
```

5.4 POST /at

Request body (JSON, at most 511 bytes; the command itself at most 255 characters):

```
{"command": "AT+GFL"}
```

Response: `data` contains the exact text the command would have printed on USB, including `\r\n` line endings and the trailing `OK` or `ERROR:<code>` (up to 8 KB). The HTTP status is `200` whenever the command was dispatched; check `data` for `ERROR`.

```
{
  "status": "success",
  "message": "Command executed",
  "data": "+FL:busy.png,png,20480\r\n+FL:free.gif,gif,51200\r\nOK\r\n"
}
```

Commands sent over HTTP are also echoed on the USB serial port, which is useful when debugging.

Commands that do not fit HTTP

Interactive commands that expect a follow-up data stream on the serial port (USB file transfer) are not usable through `/at`. Use `/file` and `/pattern` instead. Send `AT+LS` over USB (see Section 3).

5.5 GET /status

```
{
  "status": "running",
  "port": 80,
  "auth_required": false,
  "max_connections": 4,
  "wifi_mode": 3,
  "device_id": "<device id>",
  "firmware_version": "3.0",
  "project_name": "file_server",
  "compile_date": "<build date>",
  "compile_time": "<build time>"
}
```

`wifi_mode`: `0`=OFF, `1`=STA, `2`=AP, `3`=STA+AP. `auth_required` applies to all interfaces.

5.6 POST /file?name=<file>

The raw request body is written to `/data/<file>`, overwriting any existing file. The firmware frees space first if needed. On success `data` contains the file name. A partial upload is deleted.

```
curl -X POST "http://192.168.4.1/file?name=busy.png" \
  --data-binary @busy.png
```

Then display it with `AT+SP=busy.png`.

Use plain file names such as `busy.png`: no folders or path separators, no spaces and no `%`. The name is used exactly as sent (no URL decoding).

5.7 GET /file?name=<file>

Returns the file as `application/octet-stream`.

```
curl -o busy.png "http://192.168.4.1/file?name=busy.png"
```

5.8 POST /pattern

Body is plain text, one step per line, in the same format `AT+CP?` prints:

```
+CP:<led_bits>,<RRGGBB>,<speed>,<delay>
```

The existing custom pattern is replaced and saved. Invalid lines are skipped; if none are valid, `400` is returned.

```
printf '+CP:127,FF0000,5,100\n+CP:127,000000,5,100\n' > blink.txt
curl -X POST http://192.168.4.1/pattern --data-binary @blink.txt

# play it, loop forever
curl -X POST http://192.168.4.1/at -d '{"command":"AT+PP=1,255"}'
```

5.9 Status codes

Code	Meaning
200	Success (for `/at`: the command was dispatched; inspect `data`)
400	Bad request: invalid JSON, missing `command`, missing `name`, no valid pattern lines
401	Missing or wrong bearer token (only when authentication is on)
500	Any other failure. Read `message`: `File not found` (`GET /file`), `Insufficient storage space` (`POST /file`), out of memory, storage not mounted, read/write error

Table 5: HTTP status codes

The server accepts up to 4 simultaneous connections; the least recently used connection is closed when a fifth arrives.

6 Client Examples

6.1 Python

```python
import requests

BASE  = "http://192.168.1.42:8080"
TOKEN = "Q7ZpX2mWc9KdT4hLr8VbN3sYf6GjA1eU"   # only if started with AT+LS
HDRS  = {"Authorization": f"Bearer {TOKEN}"}

def at(cmd: str) -> str:
    r = requests.post(f"{BASE}/at", json={"command": cmd},
                      headers=HDRS, timeout=10)
    r.raise_for_status()
    out = r.json().get("data", "")
    if "ERROR" in out:
        raise RuntimeError(out.strip())
    return out

def upload(path: str, name: str) -> None:
    with open(path, "rb") as f:
        r = requests.post(f"{BASE}/file", params={"name": name},
                          data=f, headers=HDRS, timeout=60)
    r.raise_for_status()

upload("busy.png", "busy.png")
at("AT+SP=busy.png")
at("AT+SC=127,FF0000")
print(at("AT+GDN"))
```

6.2 PowerShell

```powershell
$base = "http://192.168.1.42:8080"
$hdrs = @{ Authorization = "Bearer Q7ZpX2mWc9KdT4hLr8VbN3sYf6GjA1eU" }

Invoke-RestMethod -Method Post -Uri "$base/at" -Headers $hdrs `
  -ContentType "application/json" -Body '{"command":"AT+SC=127,00FF00"}'

Invoke-RestMethod -Method Post -Uri "$base/file?name=free.png" `
  -Headers $hdrs -InFile .\free.png -ContentType "application/octet-stream"
```

6.3 Browser JavaScript

```javascript
const res = await fetch("http://192.168.4.1/at", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ command: "AT+SC=127,0000FF" })
});
console.log((await res.json()).data);
```

Note: a page served over `https://` cannot call the device's `http://` address (mixed-content blocking). Serve the page over plain HTTP or call the API from a native app or script.

7 Security Considerations

- Traffic is plain HTTP. The token and all data travel unencrypted, so the local API should only be used on networks you trust.

- The AP is protected by a per-device WPA2 password that is only shown on the device's own screen. Physical access to the display is effectively the credential for AP-side access.
- In AP and STA+AP modes the server runs without authentication. In STA+AP mode this also applies to the router network.
- The info page at `/` is always open and its curl examples include the current token. The token therefore does not keep out anyone who can reach the device; it only prevents accidental requests. Rely on network isolation instead.
- Keep the device on an isolated/IoT VLAN or guest network when it is used on shared infrastructure.

8 Troubleshooting

Symptom	**Check**
Connection refused in STA mode	The server stops when the device switches to STA-only mode and after every restart. Run `AT+LS?`; start it with `AT+LS=1,8080,<token>`.
`401`	Authentication is on: the token is missing, mistyped or was rotated with `AT+LT`. Read the current one with `AT+LT?` over USB.
`AT+LS` returns `ERROR`	Give all three parameters; port `1024`–`65535`; token at most 63 characters.
App cannot find the device in AP mode	The server was restarted with `AT+LS` on another port; the app expects `80`. Restart the device.
No response to `AT+LS` over HTTP	Expected: the server restarts while answering. Configure it over USB.
Connection drops after `AT+WM` over HTTP	Expected: the mode switch happens right after the response.
Settings gone after a restart	Expected in firmware 3.0: `AT+LS` and `AT+LT` settings are not stored.
Phone joins AP but has "no internet"	Expected in AP mode. Tell the phone to stay connected, or disable mobile data so requests go over Wi-Fi.
Device IP keeps changing	Create a DHCP reservation on the router.
Large uploads stall or abort	Check signal strength (`AT+STA?` RSSI). Weak links drop mid-transfer; move closer or use AP mode.
`/file` upload fails with `Insufficient storage space`	Storage full. Check `AT+GFSS`, delete files, retry.

Table 6: Troubleshooting

9 Command Quick Reference

Command	Purpose
AT+WM=<0..3>, AT+WM?	Wi-Fi mode: OFF / STA / AP / STA+AP (stored)
AT+STA=<ssid>,<pwd>, AT+STA?	Join a router / connection status
AT+GIP	Device IP on the router network
AT+AP=<ssid>,<pwd>[,...], AT+AP?	Configure / query the access point
AT+APST?	Clients connected to the AP
AT+QR=<0|1>	Hide / show the setup QR code
AT+SDLU=<url>	Base URL of the setup deep link
AT+LS=<en>,<port>,<token>, AT+LS?	Start / stop / query the HTTP server (not stored)
AT+LT?, AT+LT	Read / rotate the bearer token (not stored)

Table 7: Commands relevant to local network use