API Documentation

Backend - 500819 M1 Controller

Comprehensive API Reference

HTTP base URL: `http://<device-ip>:8080`

This reference covers the HTTP endpoints implemented by the backend. Example device and network values may differ on your unit.

Contents

1. Overview
2. Authentication and device identity
3. Device operations
4. Common errors

Overview

The server listens on port `8080`. Use the device's reachable IP address in place of `<device-ip>`, for example `http://192.168.168.52:8080`. The documentation example IP is `192.168.168.52`; actual network settings vary by device.

Send JSON request bodies with `Content-Type: application/json`. The JSON body limit is 50 MB. CORS is enabled.

Authentication and device identity

Get backend version

`GET /auth/version`

Returns the configured backend version.

Parameters: None.
Request body: None.

Success — `200 OK`

```
{
  "versionInfo": {
    "overall_version": "1.0.0"
  }
}
```

Error — `500 Internal Server Error`

```
{
  "message": "Error message"
}
```

Identify this device

`GET /auth/whoareyou`

Returns the device name, model, network details, and firmware version. The default name is `500819`.

Parameters: None.
Request body: None.

Success — `200 OK`

```
{
  "name": "500819",
  "model": "500819",
  "ipInfo": {
    "ip": "192.168.168.52",
    "netmask": "255.255.255.0",
    "gateway": "192.168.168.1",
    "ipWithCidr": "192.168.168.52/24",
    "mac": "00:11:22:33:44:55"
  },
  "firmwareVersion": "1.0.0"
}
```

Network values are read from the device and may differ from the example.

Error — `500 Internal Server Error`

```
{
  "message": "Error message"
}
```

Device operations

Reboot device

`GET /device/reboot`

Starts a device reboot after returning success.

Parameters: None.
Request body: None.

Success — `200 OK`

```
{
  "success": true
}
```

Set device name

`POST /device/setName`

Saves a device name.

Request body — required

```
{
  "deviceName": "MuxLab Controller"
}
```

Success — `200 OK`

```
{
  "success": true
}
```

Set IP settings

`POST /device/setIpSettings`

Starts network configuration with the supplied address, netmask, and gateway.

Request body — required

```
{
  "ipSettings": {
    "ip": "192.168.168.52",
    "netmask": "255.255.255.0",
    "gateway": "192.168.168.1",
    "ipWithCidr": "192.168.168.52/24",
    "mac": "00:11:22:33:44:55"
  }
}
```

`ipSettings.ip`, `netmask`, and `gateway` must be non-empty. `ipWithCidr` and `mac` are optional.

Success — `200 OK`

```
{
  "success": true
}
```

Set device date and time

`POST /device/setDateTime`

Sets the device date and time.

Request body — required

```
{
  "year": 2025,
  "month": 9,
  "day": 12,
  "hour": 14,
  "minute": 30,
  "second": 0
}
```

All six fields are required and must be number-convertible.

Success — `200 OK`

```
{
  "success": true
}
```

Invalid or missing values — `400 Bad Request`

```
{
  "message": "year, month, day, hour, minute, second are required"
}
```

Operation failure — `500 Internal Server Error`

```
{
  "message": "Error message"
}
```

Get COM port settings

`GET /device/getComPortSettings/{portNum}`

Returns saved settings for COM port A or B.

Path parameter

Name	Required	Values / behavior
`portNum`	Yes	`A` selects A; any other value selects B.

Request body: None.

Success — `200 OK`

```
{
  "mode": "RS232",
  "baudRate": 9600,
  "dataBits": 8,
  "stopBits": 1,
  "checkDigit": "NONE"
}
```

Allowed values: `mode` is `RS232`, `RS485`, or `RS422`; `baudRate` is `1200`, `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, or `115200`; `dataBits` is `7` or `8`; `stopBits` is `1` or `2`; `checkDigit` is `NONE`, `ODD`, or `EVEN`.

Settings not found — `404 Not Found`

```
{
  "message": "Com port settings not found"
}
```

Set COM port settings

`POST /device/setComPortSettings/{portNum}`

Applies and saves settings for a COM port.

Path parameter

Name	Required	Values / behavior
`portNum`	Yes	`A` selects A; any other value selects B.

Request body — required

```
{
  "comPortSettings": {
    "mode": "RS232",
    "baudRate": 9600,
    "dataBits": 8,
    "stopBits": 1,
    "checkDigit": "NONE"
  }
}
```

Use the allowed setting values listed under Get COM port settings.

Success — `200 OK`

```
{
  "success": true
}
```

Send a message over a COM port

`POST /device/sendMessageOnComPort/{portNum}`

Sends a message over the selected port and optionally returns a device response.

Path parameter

Name	Required	Values / behavior
`portNum`	Yes	`A` selects A; any other value selects B.

Request body — required

```
{
  "message": "Status?",
  "encoding": "ASCII",
  "waitForResponse": true
}
```

Field	Required	Description
`message`	Yes	Text for `ASCII`, hexadecimal data for `HEX`.
`encoding`	No	`ASCII or HEX; default to HEX.`
`waitForResponse`	No	Boolean; defaults to false.

Success — `200 OK`

```
{
  "success": true,
  "response": "OK"
}
```

Without waiting for a response, `response` is `true`. For `HEX`, a returned response is space-separated hexadecimal bytes.

Operation failure — `500 Internal Server Error`

```
{
  "message": "Error message"
}
```

Set relay state

`POST /device/setRelay/{relayNum}/{state}`

Changes and saves a relay state.

Path parameters

Name	Required	Values / behavior
`relayNum`	Yes	`A` selects A; any other value selects B.
`state`	Yes	`open` opens the relay; any other value closes it.

Request body: None.

Success — `200 OK`

```
{
  "success": true
}
```

Get relay state

`GET /device/getRelaySettings/{relayNum}`

Returns the saved state of relay A or B.

Path parameter

Name	Required	Values
`relayNum`	Yes	`A` or `B`.

Request body: None.

Success — `200 OK`

```
{
  "relayStatus": "closed"
}
```

`relayStatus` is `open` or `closed`.

Invalid relay — `400 Bad Request`

```
{
  "message": "relayNum must be A or B"
}
```

State not found — `404 Not Found`

```
{
  "message": "Relay status not found"
}
```

Operation failure — `500 Internal Server Error`

```
{
  "message": "Error message"
}
```

Get voltage-sensor input status

`GET /device/getIoPortStatus/{portNum}`

Reads the state of voltage-sensor port A or B.

Path parameter

Name	Required	Values / behavior
`portNum`	Yes	`A` selects A; any other value selects B.

Request body: None.

Success — `200 OK`

The response is a JSON string: `"open"` or `"short"`.

```
"open"
```

Learn an IR code

`GET /device/learnIRCode`

Requests one IR learning cycle.

Parameters: None.
Request body: None.

Success — `200 OK`

```
{
  "code": "Learned IR data returned by the device"
}
```

If the device times out, the response is an empty object:

```
{}
```

Emit an IR code

`POST /device/emitIRCode/{port}`

Sends an IR code on the selected port.

Path parameter

Name	Required	Values
`port`	Yes	`A` or `B`.

Request body — required

```
{
  "code": "0x38 0x00 0x69 0x22"
}
```

Success — `200 OK`

```
{
  "success": true
}
```

Operation failure — `500 Internal Server Error`

```
{
  "message": "Error message"
}
```

Common errors

Unmatched URLs return `404 Not Found`. The response includes a `message` and a `stack`; the stack value varies by environment.

```
{
  "message": "🔍 - Not Found - /unknown/path",
  "stack": "Error: 🔍 - Not Found - /unknown/path"
}
```

Some device operations return success before the hardware operation finishes. A later failure cannot change that response.