CoolMasterNet
CooLinkNet
CooLinkHub
CooLinkBridge

REST API Specification

Document Revision 0.1 Friday, December 21, 2018

Table of Contents

1 Configuration

CoolAutomation device must be properly configured to support REST API. Configuration is made via CoolAutomation's proprietary ASCII_IF protocl described in details in Programmer Reference Manual (PRM).
REST Server can be enabled and configured with `rest` command. Below example shows how to enable REST Server.

```
>rest enable
OK, Boot Required!
```

2 REST API Specification

2.1 Basics

Media Type

API relies on JSON to represent states of REST resources.

Application Root

```
/<API_version>/<Device_SN>/<Request_str>
```

- API version - Version code in format `v<X>.<Y>`
- Device_SN - CoolAutomation device serial number
- Request_str - Request string specific to API version

Examples:

```
/v1.0/device/283B96000049/raw?command=ls2
/v2.0/device/283B96002129/ls2
```

Request-URI Encoding

Request string may contain special characters that will be interpreted as shown below

Special Character	Interpretation
'&'	' ' - space
'_' - underscore	'.' - dot

For example request string "`ls&L1_100`" will be interpreted as "`ls L1.100`".

2.1.1 Error States

To indicate errors related to REST Server the common HTTP Response Status Codes are used. Additional information regarding error reason is provided in response message-body with "error" JSON key. For example:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Content-Length: 24
{"error":"Wrong Device"}
```

HTTP Response Status Code	"error" Value	Explanation
400 Bad Request	Wrong v1 route	Wrong <Request_str> format
	Wrong v2 route	
	Wrong API Version	<API Version> is not supported
403 Forbidden	Wrong Device	<Device_SN> does not match
404 Not Found	Wrong URI	URI format is wrong
405 Method Not Allowed	Wrong HTTP Method	
413 Request Entity Too Large	Fragmented HTTP header	HTTP request is fragmented
501 Not Implemented	Unsupported V2.x command	

2.2 V1 API

2.2.1 Raw Command

Request-Line

```
        GET /v1.0/device/<Device_SN>/raw?command=<CMD> HTTP/1.1
```

Response message-body

```
        {
                "command": "<CMD_interpreted>",
                "data": [
                        "<LINE1_optional>",
                        "<LINE2_optional>",
                        ...
                        "<LINEn_optional>"
                ],
                "rc": "<Exit_Code>"
        }
```

This request is used to execute CoolAutomation device ASCII_IF command. CoolAutomation's proprietary ASCII_IF protocl is described in details in Programmer Reference Manual (PRM).

Examples

- Turn ON indoor unit L1.100

Request: `/v1.0/device/283B96002128/raw?command=on&L1_100`
Response:

```
{
    "command": "on L1.100",
    "data": [],
    "rc": "OK"
}
```

- Query status of HVAC lines with ASCII_IF `line` command

Request: `/v1.0/device/283B96002128/raw?command=line`
Response:

```
{
    "command": "line",
    "data": [
        "  L1: DK Master U00/G06 myid:0B",
        "   Tx:749/749 Rx:749/749 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L2: Unused ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L3: CG5 Modbus Address:0x50(80) 9600_8N1 ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L4: Unused ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L5: Unused ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L6: Unused ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L7: Unused ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0",
        "  L8: Unused ",
        "   Tx:0/0 Rx:0/0 TO:0/0 CS:0/0 Col:0/0 NAK:0/0"
    ],
    "rc": "OK"
}
```

2.3 V2 API

2.3.1 ls, ls2

Request-Line

```
GET /v2.0/device/<Device_SN>/ls<&UID_optional> HTTP/1.1
```
or
```
GET /v2.0/device/<Device_SN>/ls2<&UID_optional> HTTP/1.1
```

Response message-body

```
{
      "command": "<CMD_interpreted>",
      "data": [
            {<JSON1_optional>},
            {<JSON2_optional>},
            ...
            {<JSONn_optional>}
      ],
      "rc": "<Exit_Code>"
}
```

This request is used to execute `ls` or `ls2` command with optional UID. Commands provide indoor unit(s) status. If UID is omitted all indoor units connected to CoolAutomation device will be listed. Response is fully JSON formatted. Response JSON keys have following meaning:

JSON Key	Meaning
"uid"	Indoor unit UID
"onoff"	ON/OFF status
"st"	Set Temperature
"rt"	Room Temperature
"fspeed"	Fan Speed
"mode"	Operation Mode
"fls"	Failure Code
"filt"	Filter Cleaning
"dmnd"	Demand (Therm_ON)

Example

Request: **`GET /v2.0/device/283B96002128/ls2&L1_100`**

Response:

```
{
    "command": "ls2 L1.100",
    "data": [
        {
            "uid": "L1.100",
            "onoff": "ON",
            "st": "062.4F",
            "rt": "076.1F",
            "fspeed": "Low",
            "mode": "Auto",
            "flr": "OK",
            "filt": "#",
            "dmnd": "0"
        }
    ],
    "rc": "OK"
}
```

3 Commands Reference

rest

3.1 rest

SYNOPSIS

```
rest
rest enable
rest disable
rest port <PORT>
```

DESCRIPTION

- Without parameters **`rest`** command displays current REST Server configuration.
- **`rest port`** command is used to configure TCP/IP listen port used by REST Server. Default port number is 10103.

EXAMPLE

```
>rest
REST            : enabled
Listen port     : 10103

>rest port 8080
OK, Boot Required!
```