GUARDIAN GVC1 BASIC JSON API

Michael Skiba

April 2018

Purpose: This document is for integrators and installers that need to send and receive basic commands to control the Guardian GVC1 Valve Controller over serial or LAN. This document assumes the integrator or installer has basic knowledge of serial communication and JSON syntax.

Revision History

Revision	Date	Author(s)	Description
0.0.1	16.04.2018	MS	Created
1.0.0	03.05.2018	MS	Added minimum firmware requirements, initial release version

Requirements

Requires a GVC1 Valve Controller with firmware version 3.4.1 or higher.

UDP Commands

Port: 9999 Local Port: 9999

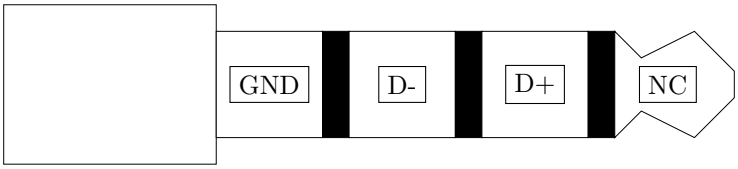
Commands can be sent over UDP to the Valve Controller. The Valve Controller is listening on port 9999. The Valve controller will reply to commands on port 9999.

RS-485 Commands

Baud Rate: 9600 Data Size:8 Parity: None

Commands can be sent over RS-485 to the Valve Controller. The connection is made with the TRRS jack near the battery compartment of the Valve Controller.

TRRS/RS-485 PINOUT



NOTE: To communicate with the Valve controller with RS-232, a RS-232 to RS-485 converter is required.

Command Syntax

NOTE: Replies are only sent after a command is received by the Valve Controller

JSON command syntax

```
{"command":"(intended command keyword)", "(command specific parameters)":(parameter value), ...,
"type":0, [silent:1]}
```

`type` - indicates the type of message
0 – request
2 – response (used on WIFI communication when replying)

`silent` - suppresses buzzer sound when receiving command, this parameter is optional
0 – not silent, same as not using the parameter
1 – silent

JSON reply syntax

```
{"command":"(received command keyword)", "(command specific parameters)":(parameter value),
...,"type":(2 or 3),"valve_id":"AABBCCDDEEFFGG"}
```

`valve_id` – unique ID of the valve that replied the command

Commands

Get Unique ID

get_unique_id: returns device unique ID

command parameters:
none

reply parameters:
”UUID”:string

command example:
`{"command":"get_unique_id","type":0}`

reply example:
`{"command":"get_unique_id","type":2,"UUID":"30AEA40322FC","valve_id":"30AEA40322FC"}`

Get Valve

get_valve: retrieves valve previously set name and location and also sensor values, connected wifi SSID, motor state and firmware version

command parameters:
none

reply parameters:
”name”: string
”location”: string
”SSID”: string
”temperature”: number
”battery”: number
”probe”: boolean

”state”: string
”version”: string

command example:
{ "command":"get_valve", "type": 0, "silent": 1}

reply example:
{"command":"get_valve","type":2,"silent":1,"name":"abc@abc.com","location":"mbl29kv73na",
"SSID":"ZWAVE","temperature":83.022995,"battery":198,"probe":true,"state":"open",
"version":"3.0.0","valve_id":"30AEA40322FC"}

Get Sensor

get_sensor: Gets information about a registered sensor.

command parameters:
”UUID”:string (contains the twelve characters representing the Unique ID(DEVICE ID) that is also printed on the product label)

reply parameters:
”UUID”:string
”battery”: number
”temperature”: number
”tilt”: boolean
”top”: boolean (last registered state of the top water sensor)
”bottom”: boolean (last registered state of the bottom water sensor)

command example:
{"command":"get_sensor", "UUID":"D88039D5D0DD","type":0}

reply example:
{"command":"get_sensor","UUID":"D88039D5D0DD","type":2,"UUID":"D88039D5D0DD","battery":100,
"temperature":69.603439,"tilt":false,"top":true,"bottom":true,"valve_id":"30AEA40322FC"}

Get Sensor List

get_sensor_list: Gets information of all registered sensors in a list.

command parameters:
none

reply parameters:
”sensors”: JSON object array

NOTE: the parameters inside each object are the same as the ones described in the ”get_sensor” command

command example:
{"command":"get_sensor_list", "type":0}

reply example:
{"command":"get_sensor_list","type":2,"sensors":[
{"UUID":"D88039D5D0DD","battery":100,"temperature":69.603439,"tilt":false,"top":true,"bottom":true},
{"UUID":"5410EC689B8C","battery":100,"temperature":67.863014,"tilt":false,"top":true,"bottom":true},
{"UUID":"5410EC687BBA","battery":83.542,"temperature":65.489716,"tilt":false,"top":true,"bottom":true},
{"UUID":"5410EC68A3E2","battery":100,"temperature":68.812340,"tilt":false,"top":true,"bottom":true},
{"UUID":"5410EC686808","battery":100,"temperature":69.761658,"tilt":false,"top":true,"bottom":true}
],"valve_id":"30AEA40322FC" }

Motor Action

motor_action: opens or closes the valve motor. This command has an acknowledgement and a reply. The acknowledgement is sent when the command is received and the motor process is started. The reply is sent when the motor process is finished.

command parameters:
"action": string (The supported values are "close" and "open")

acknowledge parameters:
"error": string (Indicates if the command was invalid. For example: trying to open when the valve is already open)

reply parameters:
"successful": number (The value is 1 if successful, 0 if the motor stalled)

command example:
{"command":"motor_action", "action":"close", "type":0}

acknowledge example:
{"command":"motor_action","action":"close","type":2,"valve_id":"30AEA40322FC"}

acknowledge error example:
{"command":"motor_action","action":"close","type":2,"error":"Command not accepted!",
"valve_id":"30AEA40322FC"}

reply example:
{"command":"motor_action","action":"close","type":2,"state":"closed","successful":1,
"valve_id":"30AEA40322FC"}

Motor State

motor_state: gets the current motor state, if open, closed or stalled.

command parameters:
none

reply parameters:
"state": string (states the current motor state. Supported strings are: "open", "opening", "openError", "closed", "closing", "closeError")

command example:
{"command":"motor_state","type":0}

reply example:
{"command":"motor_state","type":2,"state":"closed","valve_id":"30AEA402FEB8"}