DCS-400/800

Ai DCS-400 and DCS-800 4- or 8-channel controllers

Version 1.0, 01-25-2020

This is a RESTful HTTP API which uses JSON to move data to/from a target device. Rather than sending commands to set individual properties, it transfers the *state* of the device over standard HTTP methods.

This API uses the default HTTP port 80, to avoid firewall issues that may affect other ports.

Methods used

The following HTTP methods are used:

Method	Purpose
`GET`	Gets data
`POST`	Updates data
`DELETE`	Erases data/settings

Response codes

For all requests, `200 OK` indicates success with a response body (e.g. updated device limits), `204 No Content` indicates success with no body, and all other codes indicate a failure of some sort.

Paths

GET

`/info`

Returns a JSON object containing the device name, active sequence, connected lightheads, and device name.

Example requests

Gets device information

Request

No request body

Response (200 OK)

Content type: `application/json`


```
{
    "activeEvent": 0,
    "activeSequence": 0,
    "firmware": "030055_00",
    "lights": [
        "DF198-280RGBWQ4",
        ""
    ],
    "mode": "standard",
    "name": "",
    "temperatures": [
        24.9820199999999986,
        24.9229400000000005,
        24.6817899999999995,
        24.6100099999999991,
        24.7792299999999983,
        25.2210999999999999,
        25.4734499999999997,
        25.3807000000000009
    ],
    "trigger": "rising"
}
```


POST

`/info`

Route for setting device-global config state.

The following configuration settings can be set via the request body:

Setting	JSON property name	Possible values
Device name	`name`	Any ASCII string (16 characters or less)
Trigger edge	`trigger`	String: "rising" or "falling", Integer: 1 or 0, Boolean: `true` or `false`
Device mode	`mode`	"sequenced" or "standard"

A response code of `204 No Content` indicates success.

Example requests

Set name, trigger edge, and mode

Request

Content type: `application/json`

```
{
    "mode": "standard",
    "name": "Test Device",
    "trigger": "rising"
}
```

Response (204 No Content)

(No response body)

GET

`/channels`

Get channel configs. Returns JSON array containing information on the current state of all channels.

Example requests

Get Channels

Request

No request body

Response (200 OK)

Content type: `application/json`

```
[
    {
        "current": 0,
        "id": 0,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
        "id": 1,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
```

```
        "id": 2,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
        "id": 3,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
        "id": 4,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
        "id": 5,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
        "id": 6,
        "pulseDelay": 0,
        "pulseWidth": 0
    },
    {
        "current": 0,
        "id": 7,
        "pulseDelay": 0,
        "pulseWidth": 0
    }
]
```


GET

`/channels/:chan`

URL parameters

Key	Example value	Description
chan	2	Channel number [1-8]

Get individual channel data

Example requests

Get channel 2

Request

No request body

Response (200 OK)

Content type: `application/json`

```
{
    "current": 500,
    "id": 1,
    "maxCurrent": 2675,
    "pulseDelay": 0,
    "pulseWidth": 1500
}
```

POST

`/channels/:chan`

URL parameters

Key	Example value	Description
chan	7	Channel number [1-8]

Set data on an individual channel

Example requests

Set data on channel 7

Request

Content type: `application/json`

```
{
    "brightness": 57,
    "pulseDelay": 50,
    "pulseWidth": 1000
}
```

Response (204 No Content)

Content type: `text/plain`

Set data on Channel 1

Request

Content type: `application/json`

```
{
    "brightness": 57,
    "pulseDelay": 50,
    "pulseWidth": 1000
}
```

Response (204 No Content)

Content type: `text/plain`

POST

`/channels/pulse`

Pulses all channels. Acts like a hardware trigger when the device is in Standard Mode.

GET

`/events/:ev`

URL parameters

Key	Example value	Description
ev	1	Event ID [1-10]

Gets a saved Event from the controller. The `form` query in the URL can adjust how the information is returned.

The value of the `format` query can take any of the following values:

Value	Response format
`json`	JSON-formatted string
`short`	Base64 string with basic run-length encoding to save space
Anything else or absent	Non-compressed Base64 string

Example requests

Get Event information

Request

No request body

Response (200 OK)

Content type: `application/json`

```
{
    "channels": [
        {
            "current": 200,
            "id": 1,
            "mode": 1,
            "pulseDelay": 0,
            "pulseWidth": 0
        },
        {
            "current": 1250,
            "id": 3,
            "mode": 2,
            "pulseDelay": 0,
            "pulseWidth": 2500
        }
    ],
    "id": 1
}
```

POST

`/events/:ev`

URL parameters

Key	Example value	Description
ev	1	Event ID

Sets an Event by listing all applicable channel values in JSON (see the example). Channel `id`s are zero-based (0-7). All channels *not* explicitly set are zeroed-out. All channel properties (pulse width, delay, etc.) are assumed to be zero if not set explicitly.

Example requests

Set Event 1 with channels 2 and 3 (`id`s 1 and 2) in continuous mode

Request

Content type: `application/json`

```
{
    "channels": [
        {
            "current": 350,
            "id": 1,
            "mode": 1
        },
        {
            "current": 200,
            "id": 2,
            "mode": 1
        }
    ],
    "id": 1
}
```

Response (204 No Content)

(No response body)

GET

`/sequences`

Gets a JSON list of all valid Sequence IDs.

Example requests

Get Sequences

Request

No request body

Response (200 OK)

Content type: `application/json`

```
[
    1
]
```

GET

`/sequences/:seq`

URL parameters

Key	Example value	Description
seq	1	Sequence number [1-10]

Gets the events defined on a particular sequence.

Example requests

Get Sequence 1

Request

No request body

Response (200 OK)

Content type: `application/json`

```
{
    "events": [
        1,
        2
    ],
    "id": 1
}
```

POST

`/sequences/:seq`

URL parameters

Key	Example value	Description
seq	1	Sequence # [1-10]

Sets the events in a given sequence

Example requests

Set Sequence 1 to be Events 1 and 2

Request

Content type: `application/json`

```
{
    "events": [
        1,
        2
    ]
}
```

Response (204 No Content)

(No response body)

DELETE

`/sequences/:seq`

URL parameters

Key	Example value	Description
seq	1	Sequence # [1-10]

Erases a given sequence.

POST

`/stop`

Stops a running sequence, and turns off all active channels.

GET

`/network`

Get network configuration

Example requests

Get network configs

Request

No request body

Response (200 OK)

Content type: `application/json`

```
{
    "dhcp": true,
    "gateway": "192.168.100.1",
    "ipAddress": "192.168.103.22",
    "mac": "00:22:FE:03:01:AC",
    "subnetMask": "255.255.252.0"
}
```

POST

`/network`

Set network configuration. IP addresses are set in dotted-quad notation.

When setting a static IP address, the Gateway and Subnet Mask take the following default values if not set explicitly:

Property	Default value
Gateway address	IP with last octet set to 1, so IP of `192.168.1.15` yields gateway of `192.168.1.1`
Subnet mask	`255.255.255.0`

Example requests

Set network configs

Request

Content type: `application/json`

```
{
    "gateway": "10.1.100.1",
    "ipAddress": "10.1.100.4",
    "subnetMask": "255.255.255.0"
}
```

Response (200 OK)

Content type: `text/plain`

```
Set IP address to 10.1.100.4 (now static)
Set gateway IP address to 10.1.100.1
Set subnet mask to 255.255.255.0
```

DELETE

`/data`

Performs a factory reset of the device, and erases all saved configuration information.

Response is HTTP 204 (No Content).