OpenDCRE Documentation

Release 2.0

Vapor IO

March 07, 2016

Contents

1 OpenDCRE (Open Data Center Runtime Environment) 3
1.1 Introduction . . . 3
1.2 Getting OpenDCRE & OpenMistOS . . . 5
1.3 Configuring OpenDCRE & OpenMistOS . . . 8
1.4 API Reference . . . 25
1.5 OpenDCRE / OpenMistOS Release Notes . . . 40

Vapor IO Software and Hardware documentation, reference and downloads.

CHAPTER 1

OpenDCRE (Open Data Center Runtime Environment)

1.1 Introduction

OpenDCRE provides a securable RESTful API for monitoring and control of data center and IT equipment - via power line communications (PLC) over a DC bus bar, or via IPMI over LAN. The OpenDCRE API is easy to integrate into third-party monitoring, management and orchestration providers, while providing a simple, `curl`-able interface for common and custom devops tasks.

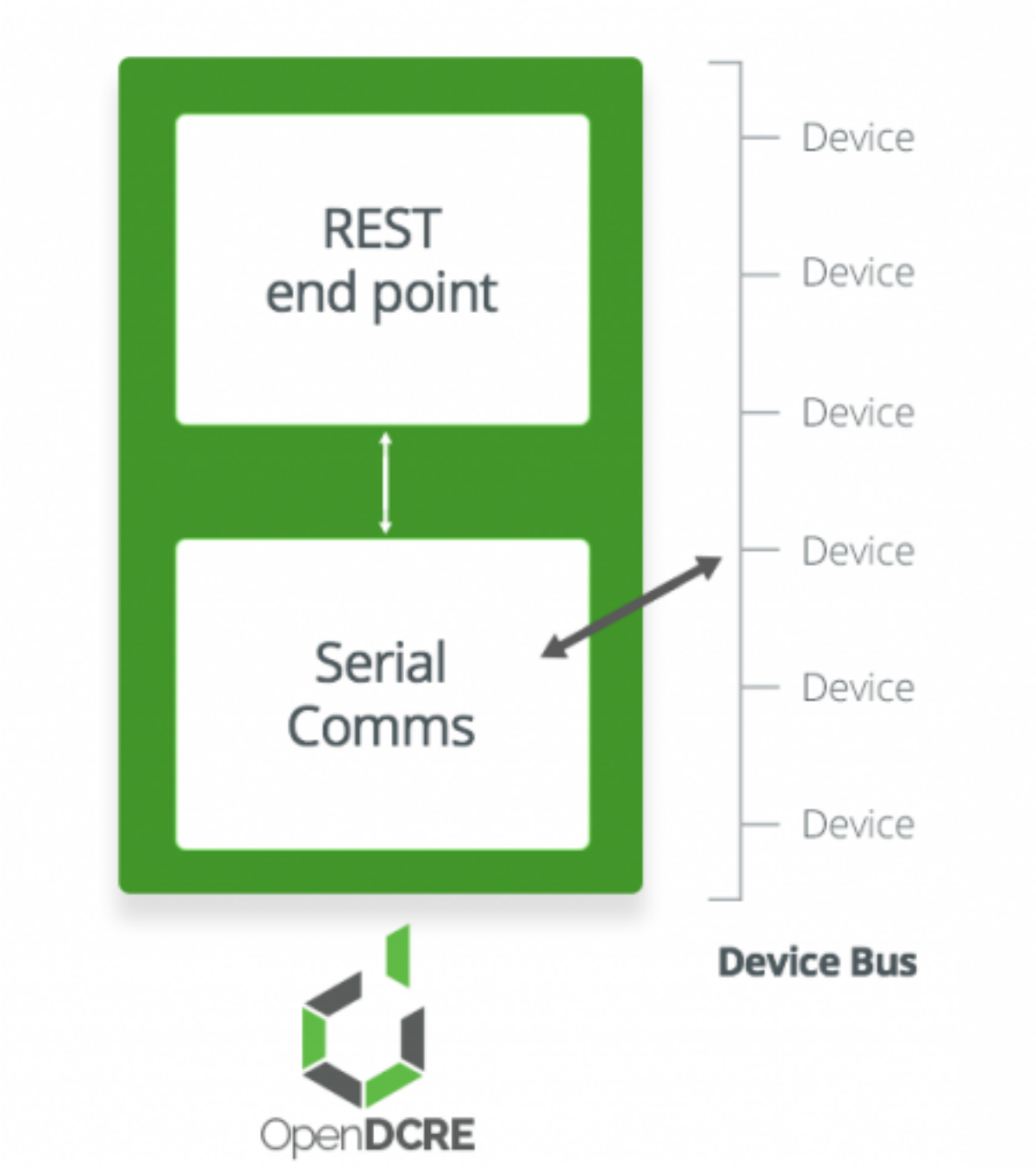
1.1.1 Features

- Simple `curl`-able RESTful API
- Analog and digital sensor support (temperature, thermistor, humidity, fan speed, pressure).
- Power control and status, including power consumption and power supply status.
- Asset information for servers.
- Physical and chassis location awareness.
- Fan speed control and status.
- Chassis "identify" LED control and status.
- System boot target selection (hdd, pxe).
- Securable via TLS/SSL.
- Integration with existing Auth providers (OAuth, LDAP, AD, etc.).
- IPMI Bridge - all OpenDCRE commands can use IPMI 2.0 over LAN as transport layer.
- Redfish support (future) - all OpenDCRE commands can use Redfish over LAN as transport layer.

1.1.2 Architecture

OpenDCRE is part of the OpenMistOS Linux distribution that runs on the Raspberry Pi 2 Model B (Raspberry Pi 3 support coming soon). OpenMistOS includes a custom Docker package (v1.10.1), compiled for ARMv7 (armhf), and OpenDCRE is packaged as a Docker container.

OpenDCRE exposes a RESTful API via an HTTP endpoint in the OpenDCRE container. The HTTP endpoint is comprised of nginx as the front-end, with uwsgi as a reverse proxy for a Python Flask application. Within Flask, OpenDCRE uses a byte-level serial protocol to communicate with the OpenDCRE device bus, which is the primary communications channel between API users and the device bus.



The OpenDCRE device bus is comprised of a set of boards and devices, individually addressable, and globally scannable for a real-time inventory of addressable devices attached to the bus. The OpenDCRE device bus allows devices to be read and written, and for various actions to be carried out, such as power control (on/off/cycle/status). Additionally, when a physical OpenDCRE device bus is not present, a software emulator can be used to simulate OpenDCRE API commands and functionality, or the included IPMI2.0 bridge may be used for IPMI communications.

All included components of OpenDCRE can be customized, integrated and secured via configuration file (nginx, uwsgi), and output their logs to a common location (/var/log/opendcre).

1.1.3 Applications

OpenMistOS and OpenDCRE can be used as an open platform for monitoring and managing data center hardware, software and environmental characteristics. Given the small form-factor of the Raspberry Pi plus its HAT board, there are a wide variety of possible applications, deployments, physical mounting strategies, and network connectivity options. Community support helps OpenDCRE grow, and enables new functionality.

1.1.4 OpenMistOS

OpenMistOS (OMOS) is an open-source operating system distribution for the Raspberry Pi, featuring OpenDCRE, the Open Data Center Runtime Environment. OMOS was developed for the purpose of using a small, single-board computer like the Raspberry Pi to perform data center sensorfication and control, particularly in concert with, but not limited to, OpenCompute-based open hardware.

OMOS v1.1.0 is Debian (jessie)-based, featuring Docker capabilities baked into the OS, with OpenDCRE running as a Dockerized service of the OS. OpenMistOS was developed by a team with decades of experience in data centers large and small, and is working to take the pain out of proprietary and arcane technologies.

1.2 Getting OpenDCRE & OpenMistOS

1.2.1 Requirements

Hardware Requirements

- Raspberry Pi 2 Model B (other Raspberry Pi versions are not supported).
- 4GB Micro SD card - 8GB or larger recommended.
- 5V Micro USB power source.
- Wired ethernet connection (wireless supported but not recommended).
- OpenDCRE HAT (optional)
- DC Bus Bar for power line communications (optional)
- IPMI 1.5-compliant BMC for IPMI bridge (optional)
- HDMI/HDMI-VGA video cable & monitor (optional)

Software Requirements

- OpenMistOS v1.1.0 or later.

1.2.2 Download and Install

Download

- Download OpenMistOS , and decompress the .img file. OpenDCRE Source on GitHub.

Install

Insert Micro SD card into card reader, and determine the SD card device:

MacOS

```
sudo diskutil list
```

Linux

```
sudo fdisk -l
```

Use `dd` to write image to card:

Macos

```
sudo dd if=<.img file> of=<sd card device> bs=4m
```

Linux

```
sudo dd if=<.img file> of=<sd card device> bs=4M
```

Note:

- `<.img file>` is the path and filename of the decompressed OpenMistOS .img downloaded above.
- `<sd card device>` is the SD card device determined in the previous step. (e.g. - /dev/disk1)

When executing the above commands, if an error is returned similar to

```
dd: <sd card device>: Resource busy
```

then the SD card must be unmounted. To do this, identify the SD card partition (can use `df -h` for this, or the results from determining the SD card device, above), then unmount the partition:

MacOS

```
sudo diskutil unmount <sd card device>
```

Linux

```
sudo umount <sd card device>
```

When `dd` is complete, OpenMistOS is ready to run from the SD card. Plug the Raspberry Pi into the wired network, insert the Micro SD card, and power up the Raspberry Pi.

At completion of the boot process, the OpenMistOS device IP address is displayed on screen (if video connection is used); alternately, check DHCP or router logs to determine the IP address of the OpenMistOS device.

Login

SSH into the OpenMistOS device:

- *Username*: `openmistos`
- *Password*: `0p3ndcr3!`

The `openmistos` user has `sudo` and Docker rights on OpenMistOS. It is recommended to **immediately** change the `openmistos` password to a new, secure, password.

Note: OpenMistOS, like other Raspberry Pi OSes, uses only the space required for the OS on the SD card. It is recommended to change this behavior so that the entire space on the SD card is used. To do this, enter the configuration menu on first login:

```
$ sudo raspi-config
```

In the configuration menu, there should be an option to use the entire disk. Once selected and confirmed, OpenMistOS will restart and the entire SD card will then be used.

Starting OpenDCRE

OpenDCRE may be started manually for verification.

To start OpenDCRE with the HAT device attached:

```
docker run -d -p 5000:5000 -v /var/log/opendcre:/logs --privileged --device /dev/mem:/dev/mem --devic
```

With Emulator

To start OpenDCRE in local emulator mode:

```
docker run -d -p 5000:5000 -v /var/log/opendcre:/logs opendcre ./start_opendcre_emulator.sh
```

Run Tests

To run the OpenDCRE test suite (from the OpenDCRE root):

```
make rpi-test
```

Verification

There are several methods for verifying that OpenDCRE is running properly.

Browser

Navigate to:

```
http://<openmistos ip address>:5000/opendcre/1.2/test
```

Output should be similar to:

```
{
    "status": "ok"
}
```

Command-Line

Running: `docker ps` produces output similar to:

```
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS
a9419ff86502        vaporio/opendcre:latest     "./start_opendcre.sh   4 days ago          Up 4 days
```

(when using the HAT)

or:

```
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS
2281101f6a60        vaporio/opendcre:latest     "./start_opendcre_em     4 days ago          Up 4 days
```

(when using the emulator)

Logs

By default, OpenDCRE logs are placed in /var/log/opendcre . Access, error and daemon logs are available for troubleshooting and analytics.

1.3 Configuring OpenDCRE & OpenMistOS

1.3.1 OpenDCRE Emulator

Emulator Configuration

In the absence of HAT hardware, OpenDCRE can utilize a software emulator to simulate and test various capabilities of OpenDCRE. The OpenDCRE emulator is configured by means of a JSON file (the default file is `simple.json`) which is stored in the `/opendcre/opendcre_southbound` directory of the OpenDCRE Docker container.

To modify the emulator output, users may clone the OpenDCRE GitHub repository, and change the contents of `simple.json`, then rebuild the OpenDCRE container.

Example emulator configuration from `simple.json` is below, followed by an explanation of the contents.

```
{
  "boards": [
    {
      "board_id": "00000001",
      "firmware_version" : "OpenDCRE Emulator v1.2.0 - Standalone Server",
      "devices" : [
        {
          "device_id": "0001",
          "device_type": "thermistor",
          "read":
            {
              "repeatable": true,
              "responses": [
                656,
                646,
                636,
                625,
                615,
                605,
                594,
                584,
                573,
                563,
                553,
                542,
                532,
                522,
                512,
                502,
                491,
```

```
            482,
            472,
            462,
            452
          ]
        }
    },
    {
      "device_id": "0002",
      "device_type": "fan_speed",
      "read":
        {
          "repeatable": true,
          "responses": [
            4100,
            4100,
            4000,
            4000,
            3900,
            3900,
            3800,
            3800,
            3700,
            3700,
            3800,
            3800,
            3900,
            3900,
            4000,
            4000,
            4100,
            4100,
            4200,
            4200
          ]
        },
      "write":
        {
          "repeatable": true,
          "responses": [
            "W1"
          ]
        }
    },
    {
      "device_id": "0004",
      "device_type": "system",
      "asset_info":
      {
          "repeatable": true,
          "responses": [
            "not yet implemented"
          ]
      },
      "boot_target": {
          "repeatable": true,
          "responses": [
            "not yet implemented"
```

```
            ]
        }
      },
      {
        "device_id": "0005",
        "device_type": "led",
        "read":
          {
            "repeatable": true,
            "responses": [
              1,
              0
            ]
          },
        "write":
          {
            "repeatable": true,
            "responses": [
              "W1"
            ]
          }
      },
      {
        "device_id": "0009",
        "device_type": "thermistor",
        "read":
          {
            "repeatable": true,
            "responses": [
              656,
              646,
              636,
              625,
              615,
              605,
              594,
              584,
              573,
              563,
              553,
              542,
              532,
              522,
              512,
              502,
              491,
              482,
              472,
              462,
              452
            ]
          }
      },
      {
        "device_id": "2000",
        "device_type": "temperature",
        "read":
          {
```

```
            "repeatable": true,
            "responses": [
              28.78,
              29.77,
              30.75,
              31.84,
              32.82,
              33.81,
              34.89,
              35.88,
              36.96,
              37.94,
              38.93,
              40.21,
              41.27,
              42.33,
              43.39,
              44.45,
              45.61,
              46.57,
              47.63,
              48.69,
              49.75
            ]
          }
        },
        {
          "device_id": "4000",
          "device_type": "temperature",
          "read":
            {
              "repeatable": true,
              "responses": [
                28.78,
                29.77,
                30.75,
                31.84,
                32.82,
                33.81,
                34.89,
                35.88,
                36.96,
                37.94,
                38.93,
                40.21,
                41.27,
                42.33,
                43.39,
                44.45,
                45.61,
                46.57,
                47.63,
                48.69,
                49.75
              ]
            }
        },
        {
```

```
          "device_id": "000D",
          "device_type": "power",
          "power":
            {
              "repeatable": true,
              "responses": [
                "0,0,0,0"
              ]
            }
        }
      ]
    },
    {
      "board_id": "00000002",
      "firmware_version" : "OpenDCRE Emulator v1.2.0 - Microserver",
      "devices" : [
        {
          "device_id": "0002",
          "device_type": "fan_speed",
          "read":
            {
              "repeatable": true,
              "responses": [
                4100,
                4100,
                4000,
                4000,
                3900,
                3900,
                3800,
                3800,
                3700,
                3700,
                3800,
                3800,
                3900,
                3900,
                4000,
                4000,
                4100,
                4100,
                4200,
                4200
              ]
            },
          "write":
            {
              "repeatable": true,
              "responses": [
                "W1"
              ]
            }
        },
        {
          "device_id": "8001",
          "device_type": "system",
          "asset_info":
          {
```

```
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        },
        "boot_target": {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        }
      },
      {
        "device_id": "8002",
        "device_type": "power",
        "power":
          {
            "repeatable": true,
            "responses": [
              "0,0,0,0"
            ]
          }
      },
      {
        "device_id": "8003",
        "device_type": "temperature",
        "read":
          {
            "repeatable": true,
            "responses": [
              28.78,
              29.77,
              30.75,
              31.84,
              32.82,
              33.81,
              34.89,
              35.88,
              36.96,
              37.94,
              38.93,
              40.21,
              41.27,
              42.33,
              43.39,
              44.45,
              45.61,
              46.57,
              47.63,
              48.69,
              49.75
            ]
          }
      },
      {
        "device_id": "8101",
        "device_type": "system",
        "asset_info":
```

```
        {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        },
        "boot_target": {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        }
      },
      {
        "device_id": "8102",
        "device_type": "power",
        "power":
          {
            "repeatable": true,
            "responses": [
              "0,0,0,0"
            ]
          }
      },
      {
        "device_id": "8103",
        "device_type": "temperature",
        "read":
          {
            "repeatable": true,
            "responses": [
              28.78,
              29.77,
              30.75,
              31.84,
              32.82,
              33.81,
              34.89,
              35.88,
              36.96,
              37.94,
              38.93,
              40.21,
              41.27,
              42.33,
              43.39,
              44.45,
              45.61,
              46.57,
              47.63,
              48.69,
              49.75
            ]
          }
      },
      {
        "device_id": "8201",
        "device_type": "system",
```

```
        "asset_info":
        {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        },
        "boot_target": {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        }
      },
      {
        "device_id": "8202",
        "device_type": "power",
        "power":
          {
            "repeatable": true,
            "responses": [
              "0,0,0,0"
            ]
          }
      },
      {
        "device_id": "8203",
        "device_type": "temperature",
        "read":
          {
            "repeatable": true,
            "responses": [
              28.78,
              29.77,
              30.75,
              31.84,
              32.82,
              33.81,
              34.89,
              35.88,
              36.96,
              37.94,
              38.93,
              40.21,
              41.27,
              42.33,
              43.39,
              44.45,
              45.61,
              46.57,
              47.63,
              48.69,
              49.75
            ]
          }
      },
      {
        "device_id": "8301",
```

```
        "device_type": "system",
        "asset_info":
        {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        },
        "boot_target": {
            "repeatable": true,
            "responses": [
              "not yet implemented"
            ]
        }
      },
      {
        "device_id": "8302",
        "device_type": "power",
        "power":
          {
            "repeatable": true,
            "responses": [
              "0,0,0,0"
            ]
          }
      },
      {
        "device_id": "8303",
        "device_type": "temperature",
        "read":
          {
            "repeatable": true,
            "responses": [
              28.78,
              29.77,
              30.75,
              31.84,
              32.82,
              33.81,
              34.89,
              35.88,
              36.96,
              37.94,
              38.93,
              40.21,
              41.27,
              42.33,
              43.39,
              44.45,
              45.61,
              46.57,
              47.63,
              48.69,
              49.75
            ]
          }
      },
      {
```

```
      "device_id": "0005",
      "device_type": "led",
      "read":
        {
          "repeatable": true,
          "responses": [
            1,
            0
          ]
        },
      "write":
        {
          "repeatable": true,
          "responses": [
            "W1"
          ]
        }
    },
    {
      "device_id": "2000",
      "device_type": "temperature",
      "read":
        {
          "repeatable": true,
          "responses": [
            28.78,
            29.77,
            30.75,
            31.84,
            32.82,
            33.81,
            34.89,
            35.88,
            36.96,
            37.94,
            38.93,
            40.21,
            41.27,
            42.33,
            43.39,
            44.45,
            45.61,
            46.57,
            47.63,
            48.69,
            49.75
          ]
        }
    },
    {
      "device_id": "4000",
      "device_type": "temperature",
      "read":
        {
          "repeatable": true,
          "responses": [
            28.78,
            29.77,
```

```
                30.75,
                31.84,
                32.82,
                33.81,
                34.89,
                35.88,
                36.96,
                37.94,
                38.93,
                40.21,
                41.27,
                42.33,
                43.39,
                44.45,
                45.61,
                46.57,
                47.63,
                48.69,
                49.75
              ]
            }
        },
        {
          "device_id": "000D",
          "device_type": "power",
          "power":
            {
              "repeatable": true,
              "responses": [
                "0,0,0,0"
              ]
            }
        }
      ]
    }
  ]
}
```

The OpenDCRE emulator simulates two different boards (servers) - the first being a single-node server, and the second a multi-node (microserver). The JSON document file is structured around a collection of boards and devices.

Boards

Each board must have a `board_id` and `firmware_version` field. Each `board_id` must be a unique 4-byte value, encoded as a hex string between "00000000" and "00FFFFFF" (the upper byte is reserved, and must always be 00), and `firmware_version` must be a string value (including empty string).

The `board_id` is used in the OpenDCRE API to address a given board, while the `firmware_version` field is used to populate the `firmware_version` field of the response to the OpenDCRE "version" command for a given board (e.g.

```
http://<ipaddress>:5000/opendcre/1.2/version/1
```

gets the version information for board 1).

As with all commands in OpenDCRE, if a board or device does not exist in the emulator configuration, then a 500 error is returned as the result of a given command.

Devices

A given board also has a collection of devices. Each device is identified by a `device_id`, used to indicate a given device in an OpenDCRE command - e.g.:

```
http://<ipaddress>:5000/opendcre/1.2/read/thermistor/00000001/0001
```

The `device_id` field is a 2-byte value represented as a hexadecimal string that is unique to a given board.

Device Types

The `device_type` field must be present, and must contain a string value that corresponds to an OpenDCRE-supported device type. This list includes:

- `thermistor`
- `power`
- `humidity`
- `pressure` (not implemented)
- `led`
- `system`
- `fan_speed`
- `temperature`

Changed in version 1.2: In previous releases, a device type of `none` indicated that no device is present at a given `device_id` on the given board, and may be ignored. In OpenDCRE v1.2 the `none` device type has been removed.

Other device types (e.g. for additional sensors and actions) will be added in future revisions of OpenDCRE, or may be added by developers wishing to add support for other device types.

Finally, a field corresponding to the action supported for a given device type is required. A map of device types to supported actions is below:

Device Type	Action Supported
`thermistor`	`read`
`temperature`	`read`
`power`	`power`
`humidity`	`read`
`pressure`	not supported yet
`led`	`read`, `write`
`fan_speed`	`read`, `write`
`system`	not supported yet

Read

For the `read` action's field in the OpenDCRE emulator configuration, two fields may be configured relating to the responses returned from a read command for the given device.

First, the `repeatable` field may be set to true or false, depending on whether it is desirable for the list of responses set in the responses field to repeat in a round-robin fashion, or if a device should stop returning data after its response list has been exhausted.

The `responses` field is a list of zero or more values that may be returned for a given read command. The raw values are converted (where necessary) by the built-in OpenDCRE conversion functions, based on the given `device_type`. Some examples are given for the thermistor sensor device type in the `simple.json` file.

When a list of values is provided for responses, the emulator iterates sequentially through the items in that list, until the list is exhausted (if repeatable is set to "true", then the emulator returns to the beginning of the list).

An empty responses list means the device returns no data, which translates to a 500 error for the read command at the OpenDCRE REST API level (useful for simulating errors). To always return the same single value, a responses list with a single element, and repeatable set to "true" will suffice.

Read Response Format

The table below describes the response format for each device type for `read` commands to the emulator.

Device Type	Format
`thermistor`	integer, converted by OpenDCRE (see `simple.json`)
`temperature`	numeric, sent back as numeric value (e.g. 28.78)
`humidity`	numeric, converted by OpenDCRE
`led`	integer, `1` is `on` and `0` is `off`; all other values are errors
`fan_speed`	integer, sent back as integer value (e.g. 4100)

Values that do not conform to the above formats will result in errors to `read` requests made to the emulator, as they would on the device bus.

Write

For the `write` action's field in the OpenDCRE emulator configuration, two fields may be configured, relating to the responses returned from a write command for the given device. The fields are laid out and function in the same manner as `read` fields.

Write Response Format

The table below describes the response format for each device type for `write` commands to the emulator.

Device Type	Format
`led`	string - `W1` is successful, while `W0` is unsuccessful; all other values are errors.
`fan_speed`	string - `W1` is successful, while `W0` is unsuccessful; all other values are errors.

Values that do not conform to the above formats will result in errors to `write` requests made to the emulator, as they would on the device bus.

Writing to a device from OpenDCRE to the emulator does not currently result in any state change for a corresponding device in the emulator. That functionality may be added in a future release.

Power

For the `power` action's field in the OpenDCRE emulator configuration, similar fields are present - repeatable and responses.

For every power command (e.g. `on`/`off`/`cycle`/`status`) issued to a power device in the OpenDCRE emulator, a response is returned from the responses list, which may be repeatable or non-repeatable. The values in the responses list correspond to power status values returned over PMBUS from the hot swap controller on an OCP server, and are expressed as an integer value in the emulator configuration (see example above). OpenDCRE converts the raw response to a friendly power status result using its built-in conversion functions.

Other Notes

The emulator configuration in `simple.json` is designed to provide a simple view and demonstration of how OpenDCRE works. The OpenDCRE emulator is also used for testing purposes, and additional emulator configurations may be found under the `/opendcre/opendcre_southbound/tests/data` directory of the OpenDCRE Docker container.

An invalid emulator configuration will cause the OpenDCRE emulator to fail to start or function properly.

Additional features of the emulator that may be used by advanced users or hardware/protocol developers include:

- Ability to send back raw bytes for responses to `scan`, `version`, `read`, `write`, and `power` commands. In tests, this can be seen where a list (or list of lists) of integer values is specified for a given response. Special sentinel values (999, 10xx) are used to place sequence numbers and checksums into the packet stream.
- Ability to support command retries in cases of invalid packets, line noise, etc.
- Ability to support 'scan-all' command and retries using time-division multiplexing; success and failure scenarios may be implemented for various configurations. See the `test-scanall` tests.
- IPMI emulator support is not yet included, but may be in a future release.

1.3.2 IPMI Bridge

About

The Vapor IPMI bridge allows users of OpenDCRE to utilize both bus-bar-based power line communications, and LAN-based IPMI communications for equipment monitoring and management. The IPMI bridge is included with OpenDCRE v1.1.0 and later, and supports power control and status via IPMI using the OpenDCRE REST API.

Requirements

- OpenMistOS must be connected to a wired LAN network that can reach all BMCs configured to be managed over OpenDCRE.
- Knowledge of BMC IP addresses, authentication, integrity and encryption types as well as usernames and passwords (where applicable) required.
- **IPMI 2.0 Authentication types supported:**
  - `NO_AUTHENTICATION_ALGORITHM` : No authentication used to establish IPMI session.
  - `RAKP_HMAC_SHA1` : RAKP HMAC SHA1-128 authentication used to establish IPMI session.
- **IPMI 2.0 Integrity types supported:**
  - `NO_INTEGRITY_ALGORITHM` : No integrity algorithm used in IPMI session.
  - `HMAC_SHA1_96` : HMAC SHA1-96 used to verify packet integrity.
- **IPMI 2.0 Encryption types supported:**
  - `NO_ENCRYPTION_ALGORITHM` : No encryption used to ensure confidentiality of IPMI packets in session.
  - `AES_CBC_128` : AES 128-bit CBC encryption used to ensure confidentiality of IPMI packets.

Configuration

The Vapor IPMI bridge is configured via the `bmc_config.json` file, which must be placed in the top level of the OpenDCRE distribution. An example file, `bmc_config_sample.json` is included with OpenDCRE (located in the `opendcre_southbound` directory), and may be modified to one's environment.

All IPMI BMCs successfully configured will show up on a `scan` command result as devices under `board_id` 40000000.

```
{
  "bmcs": [
    {
      "bmc_ip": "192.168.1.118",
      "username": "ADMIN",
      "password": "ADMIN",
      "auth_type": "RAKP_HMAC_SHA1",
      "integrity_type": "HMAC_SHA1_96",
      "encryption_type": "AES_CBC_128"
    },
    {
      "bmc_ip": "192.168.1.119",
      "username": "root",
      "password": "vapor",
      "auth_type": "RAKP_HMAC_SHA1",
      "integrity_type": "HMAC_SHA1_96",
      "encryption_type": "AES_CBC_128"
    }
  ]
}
```

For each BMC supported, an entry is added to the `bmcs` list above. Each entry must include:

- `bmc_ip` - the IP address (as a string) corresponding to the BMC to be managed. IP address must be reachable by OpenMistOS.
- `username` - the username to use in connecting to the BMC - may be an empty string if no username is used.
- `password` - the password to use in connecting to the BMC - may be an empty string if no username is used.
- **`auth_type` - the type of authentication to use in connecting to the BMC, supported values:**
  - `NO_AUTHENTICATION_ALGORITHM`
  - `RAKP_HMAC_SHA1`
- **`integrity_type` - the type of integrity validation to use in communicating with the BMC, supported values:**
  - `NO_INTEGRITY_ALGORITHM`
  - `HMAC_SHA1_96`
- **`encryption_type` - the type of encryption to use in communicating with the BMC, supported values:**
  - `NO_ENCRYPTION_ALGORITHM`
  - `AES_CBC_128`

If a field is missing, or the `bmc_config.json` file is improperly formatted, OpenDCRE IPMI capabilities will not be available.

Once the configuration file has been successfully edited, rebuild the OpenDCRE Docker container, and verify the configured BMC devices are returned via a `scan` command.

Each BMC device will show up as a board, with `board_id` in the range of `40000001` ... `40FFFFFF`.

OpenDCRE commands may be issued against IPMI and PLC devices without change in command format.

Tested BMCs

OpenDCRE v1.2 has been tested and verified to be compatible with IPMI 2.0 connections and commands for the following BMCs:

- ASpeed AST2400 (via HPE CL7100)
- Nuvoton WPCM450RA0BK (via SuperMicro X7SPA-HF)
- ASpeed AST2050 (via Tyan S8812)
- ASpeed AST1250 (via Freedom)

The OpenDCRE community welcomes testing and bug reports against other BMCs and system types.

1.3.3 Configuring OpenDCRE

Customization

OpenDCRE may be customized in a variety of ways, most commonly by changing the HTTP endpoint port, adding TLS certificates for HTTPS support, or by integrating OpenDCRE with a site-specific authentication provider.

Port

To change the port OpenDCRE listens on, edit the `opendcre_nginx.conf` file, and rebuild the OpenDCRE docker container. Be sure to also update the `DOCKER_RUN` variable in the `/etc/init.d/opendcre` init.d script to indicate the correct port mapping for the Docker container, as well.

```
server {
    listen 5000;
    server_name localhost;
    charset utf-8;
    access_log /logs/opendcre.net_access.log;
    error_log /logs/opendcre.net_error.log;

    location / {
        add_header 'Access-Control-Allow-Origin' '*';
        uwsgi_pass unix://var/uwsgi/opendcre.sock;
        include /etc/nginx/uwsgi_params;
    }
}
```

TLS/SSL

TLS/SSL certificates may be added to OpenDCRE via Nginx configuration. Refer to Nginx documentation for instructions on how to enable TLS.

Authentication

As OpenDCRE uses Nginx as its reverse proxy, authentication may be enabled via Nginx configuration - see Nginx documentation for instructions on how to enable authentication.

1.3.4 Running and Testing OpenDCRE

When starting OpenDCRE manually, the following steps may be followed.

- First, OpenDCRE expects a volume to be exposed for logs (`/logs` is the location within the container, which should be mapped externally).
- Additionally, OpenDCRE, by default, uses TCP port 5000 to listen for API requests.
- In cases where the OpenDCRE HAT is used with the OpenDCRE container, the `/dev/ttyAMA0` serial device is also required.
- In cases where the OpenDCRE HAT is used with the OpenDCRE container, `/dev/mem` must also be provided to the container for use by RPI GPIO for modem configuration.
- Finally, in cases where the HAT is used with OpenDCRE, the container must also be set to `--privileged` to allow GPIO access.

With HAT

To start OpenDCRE with the HAT device attached:

```
docker run -d -p 5000:5000 -v /var/log/opendcre:/logs --privileged --device /dev/mem:/dev/mem --devic
```

With Emulator

To start OpenDCRE in local emulator mode:

```
docker run -d -p 5000:5000 -v /var/log/opendcre:/logs opendcre ./start_opendcre_emulator.sh
```

Run Tests

To run the OpenDCRE test suite (from OpenDCRE root directory):

```
make rpi-test
```

1.3.5 Building OpenDCRE Docker Container

Building a modified OpenDCRE container is a fairly straightforward process. First, clone the OpenDCRE GitHub repository to a location on OpenMistOS.

Next, to build a custom distribution of OpenDCRE (for example, to include site-specific TLS certificates, IPMI BMC configuration, or to configure nginx to use site-specific authn/authz), the included Dockerfile can be used to package up the distribution.

In the simplest case, from the opendcre directory:

```
docker build -t opendcre:custom-v1.2.0 -f Dockerfile.rpi .
```

Apply whatever tag is most descriptive for the custom image.

From this point, test and run OpenDCRE to ensure the changes were successful.

1.3.6 Updating OpenDCRE

OpenDCRE Updates

In OpenMistOS, upgrades to OpenDCRE may be carried out by updating the OpenDCRE docker container. To do this, first stop OpenDCRE.

Then, log in to Docker Hub (assumes OpenMistOS has Internet access):

```
$ docker login
```

(enter your Docker Hub username, password and email address)

```
$ docker pull vaporio/opendcre
```

If an update is available, the latest version of opendcre will be pulled down to OpenMistOS.

Finally, start OpenDCRE again.

OpenMistOS Updates

To update OpenMistOS:

```
$ sudo apt-get update && sudo apt-get upgrade
```

1.4 API Reference

The examples below assume OpenDCRE is running on a given <ipaddress> and <port>. The default port for OpenDCRE is TCP port 5000. Currently, all commands are GET requests; a future version will expose these commands via POST as well.

1.4.1 Scan

Description

The `scan` command polls boards and devices attached to the board. The `scan` command takes a `board_id` as its argument, and returns an array of board and device descriptors. If no `board_id` is provided, the `scan` command scans all boards on the device bus.

Note: It is likely a good idea for applications to scan for all boards on startup, to ensure a proper map of boards and devices is available to the application. Mismatches of board and device types and identifiers will result in 500 errors being returned for various commands that rely on these values mapping to actual hardware.

Request Format

Scan devices on a specific `board_id`:

```
http://<ipaddress>:<port>/opendcre/<version>/scan/<board_id>
```

Scan all boards on the device bus:

```
http://<ipaddress>:<port>/opendcre/<version>/scan
```

Parameters

board_id (optional) Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge boards have a special `board_id` of 40NNNNNN (where NNNNNN is the hex string id of each configured BMC).

Request Example

```
http://opendcre:5000/opendcre/1.2/scan
```

Response Schema

```
{
    "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-boards-devices",
    "title": "OpenDCRE Boards and Devices",
    "type": "object",
    "properties": {
        "boards": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "board_index": {
                        "type": "string"
                    },
                    "devices": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "device_id": {
                                    "type": "string"
                                },
                                "device_type": {
                                    "type": "string",
                                    "enum": [
                                        "temperature",
                                        "thermistor",
                                        "humidity",
                                        "led",
                                        "system",
                                        "power",
                                        "fan_speed",
                                        "pressure"
                                    ]
                                }
                            }
                        }
                    }
                }
            }
```

```
        }
    }
}
```

Example Response

```
{
  "boards": [
    {
      "board_id": "00000001",
      "devices": [
        {
          "device_id": "0001",
          "device_type": "thermistor"
        },
        {
          "device_id": "0002",
          "device_type": "fan_speed"
        }
      ]
    },
    {
      "board_id": "00000002",
      "devices": [
        {
          "device_id": "0001",
          "sensor_type": "thermistor"
        },
        {
          "device_id": "2000",
          "device_type": "temperature"
        }
      ]
    }
  ]
}
```

Errors

Returns error (500) if scan command fails, or if `board_id` corresponds to an invalid `board_id`.

1.4.2 Version

Description

Return version information about a given board given its `board_id`.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/version/<board_id>
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40000000.

Request Example

```
http://opendcre:5000/opendcre/1.2/version/00000001
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-version",
  "title": "OpenDCRE Board Version",
  "type": "object",
  "properties": {
    "api_version": {
      "type": "string"
    },
    "firmware_version": {
      "type": "string"
    },
    "opendcre_version": {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "api_version": "1.2",
  "firmware_version": "OpenDCRE Emulator v1.2.0",
  "opendcre_version": "1.2.0"
}
```

Errors

Returns error (500) if version retrieval does not work or if `board_id` specifies a nonexistent board.

1.4.3 Read Device

Description

Read a value from the given `board_id` and `device_id` for a specific `device_type`. The specified `device_type` must match the actual physical device type (as reported by the `scan` command), and is used to return a translated raw reading value (e.g. temperature in C for a thermistor) based on the existing algorithm for a given sensor type. The raw value is also returned.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/read/<device_type>/<board_id>/<device_id>
```

Parameters

device_type String value (lower-case) indicating what type of device to read: `thermistor`, `temperature`, `humidity`, `led`, `fan_speed`, `pressure` (not implemented yet)

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN (where NNNNNN is the hex string id of each individual BMC configured with the IPMI Bridge).

device_id The device to read on the specified board. Hexadecimal string representation of a 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE matches the `device_type` specified in the command for the given device - else, a 500 error is returned.

Request Example

```
http://opendcre:5000/opendcre/1.2/read/thermistor/00000001/0001
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-thermistor-reading",
  "title": "OpenDCRE Thermistor Reading",
  "type": "object",
  "properties": {
    "temperature_c": {
      "type": "number"
    }
  }
}
```

Example Response

```
{
  "temperature_c": 19.73
}
```

Errors

If a device is not readable or does not exist, an error (500) is returned.

1.4.4 Get Asset Information

Description

Get asset information from the given `board_id` and `device_id`. The device's `device_type` must be of type `system` (as reported by the `scan` command), and is used to return asset information for a given device.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/asset/<board_id>/<device_id>
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN, where NNNNNN corresponds to the hex string id of each configured BMC.

device_id The device to read asset information for on the specified board. Hexadecimal string representation of a 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE is of type `system` - else, a 500 error is returned.

Request Example

```
http://opendcre:5000/opendcre/1.2/asset/00000001/0004
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-asset-information",
  "title": "OpenDCRE Asset Information",
  "type": "object",
  "properties": {
    "bmc_ip": {
      "type": "string"
    },
    "board_info": {
      "type": "object",
      "properties": {
        "manufacturer": {
          "type": "string"
        },
        "part_number": {
          "type": "string"
        },
        "product_name": {
          "type": "string"
        },
        "serial_number": {
          "type": "string"
        }
      }
```

```
    },
    "chassis_info": {
      "type": "object",
      "properties": {
        "chassis_type": {
          "type": "string"
        },
        "part_number": {
          "type": "string"
        },
        "serial_number": {
          "type": "string"
        }
      }
    },
    "product_info": {
      "type": "object",
      "properties": {
        "asset_tag": {
          "type": "string"
        },
        "manufacturer": {
          "type": "string"
        }
        "part_number": {
          "type": "string"
        },
        "product_name": {
          "type": "string"
        },
        "serial_number": {
          "type": "string"
        },
        "version": {
          "type": "string"
        }
      }
    }
  }
}
```

Example Response

```
{
  "bmc_ip": "192.168.1.118",
  "board_info": {
    "manufacturer": "Vapor IO",
    "part_number": "0001",
    "product_name": "Example Product",
    "serial_number": "S1234567"
  },
  "chassis_info": {
    "chassis_type": "rack mount chassis",
    "part_number": "P1234567",
    "serial_number": "S1234567"
  },
  "product_info": {
```

```
    "asset_tag": "A1234567",
    "manufacturer": "Vapor IO",
    "part_number": "P1234567",
    "product_name": "Example Product",
    "serial_number": S1234567",
    "version": "v1.2.0"
  }
}
```

Errors

If asset info is unavailable or does not exist, an error (500) is returned.

1.4.5 Power

Description

Control device power, and/or retrieve its power supply status.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/power/<board_id>/<device_id>[/<command>]
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN, where NNNNNN corresponds to the hex string id of each configured BMC.

device_id The device to issue power command to on the specified board. Hexadecimal string representation of 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE is `power` - else, a 500 error is returned.

command (optional) `on` : Turn power on to specified device. `off` : Turn power off to specified device. `cycle` : Power-cycle the specified device. `status` : Get power status for the specified device.

For all commands, power status is returned as the command's response.

Request Example

```
http://opendcre:5000/opendcre/1.2/power/00000001/000d/on
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-power-status",
  "title": "OpenDCRE Power Status",
  "type": "object",
  "properties": {
```

```
    "input_power": {
      "type": "number"
    },
    "input_voltage": {
      "type": "number"
    },
    "output_current": {
      "type": "number"
    },
    "over_current": {
      "type": "boolean"
    },
    "pmbus_raw": {
      "type": "string"
    },
    "power_ok": {
      "type": "boolean"
    },
    "power_status": {
      "type": "string"
    },
    "under_voltage": {
      "type": "boolean"
    }
  }
}
```

Example Response

```
{
  "input_power": 0.0,
  "input_voltage": 0.0,
  "output_current": -25.70631970260223,
  "over_current": false,
  "pmbus_raw": "0,0,0,0",
  "power_ok": true,
  "power_status": "on",
  "under_voltage": false
}
```

Errors

If a power action fails, or an invalid board/device combination are specified, an error (500) is returned.

1.4.6 Boot Target

Description

The boot target command may be used to get or set the boot target for a given device (whose device_type must be `system`). The boot_target command takes two required parameters - `board_id` and `device_id`, to identify the device to direct the boot_target command to. Additionally, a third, optional parameter, `target` may be used to set the boot target.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/boot_target/<board_id>/<device_id>[/<target>]
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN, where NNNNNN corresponds to the hex string id of each configured BMC.

device_id The device to issue boot target command to on the specified board. Hexadecimal string representation of 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE is `system` - else, a 500 error is returned.

target (optional) `hdd` : boot to hard disk. `pxe` : boot to network. `no_override` : use the system default boot target.

If a target is not specified, boot_target makes no changes, and simply retrieves and returns the system boot target. If `target` is specified and valid, the boot_target command will return the updated boot target value, as provided by the remote device.

Request Example

```
http://opendcre:5000/opendcre/1.2/boot_target/00000001/0004
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-boot-target",
  "title": "OpenDCRE Boot Target",
  "type": "object",
  "properties": {
    "target": {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "target": "no_override"
}
```

Errors

If a boot target action fails, or an invalid board/device combination are specified, an error (500) is returned.

1.4.7 Location

Description

The location command returns the physical location of a given board in the rack, if known, and may also include a given device's position within a chassis (when `device_id` is specified). IPMI boards return `unknown` for all fields of `physical_location` as location information is not provided by IPMI.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/location/<board_id>[/<device_id>]
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN, where NNNNNN corresponds to the hex string id of each configured BMC.

device_id (optional) The device to get location for on the specified board. Hexadecimal string representation of 2-byte integer value - range 0000..FFFF. Must be a valid, existing device known to OpenDCRE - else, a 500 error is returned.

Response Schema

Device Location

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-device-location",
  "title": "OpenDCRE Device Location",
  "type": "object",
  "properties": {
    "chassis_location": {
      "type": "object",
      "properties": {
        "depth": {
          "type": "string"
        },
        "horiz_pos": {
          "type": "string"
        },
        "vert_pos": {
          "type": "string"
        },
        "server_node": {
          "type": "string"
        }
      }
    },
    "physical_location": {
      "type": "object",
      "properties": {
        "depth": {
          "type": "string"
```

```
            },
            "horizontal": {
              "type": "string"
            },
            "vertical": {
              "type": "string"
            }
          }
        }
      }
    }
```

Board Location

```
    {
      "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-board-location",
      "title": "OpenDCRE BoardLocation",
      "type": "object",
      "properties": {
        "physical_location": {
          "type": "object",
          "properties": {
            "depth": {
              "type": "string"
            },
            "horizontal": {
              "type": "string"
            },
            "vertical": {
              "type": "string"
            }
          }
        }
      }
    }
```

Example Responses

Device Location

```
    {
      "chassis_location": {
        "depth": "unknown",
        "horiz_pos": "unknown",
        "server_node": "unknown",
        "vert_pos": "unknown"
      },
      "physical_location": {
        "depth": "unknown",
        "horizontal": "unknown",
        "vertical": "unknown"
      }
    }
```

- Valid values for `chassis_location` `depth` fields are `front`, `middle` and `rear`.
- Valid values for `chassis_location` `horiz_pos` fields are `left`, `middle` and `right`.

- Valid values for `chassis_location` `vert_pos` fields are `top`, `middle`, and `bottom`.
- `unknown` is a valid value for any location field.

Board Location

```
{
  "physical_location": {
    "depth": "unknown",
    "horizontal": "unknown",
    "vertical": "unknown"
  }
}
```

- Valid values for `physical_location` `depth` fields are: `front`, `middle`, and `rear`.
- Valid values for `physical_location` `horizontal` fields are: `left`, `middle`, and `right`.
- Valid values for `physical_location` `vertical` fields are: `top`, `middle`, and `bottom`.
- `unknown` is a valid value for any location field.

Errors

If a location command fails, or an invalid board/device combination are specified, an error (500) is returned.

1.4.8 LED Control

Description

The LED control command is used to get and set the chassis "identify" LED state. `led` devices known to OpenDCRE allow LED state to be set and retrieved.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/led/<board_id>/<device_id>[/<led_state>]
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN, where NNNNNN corresponds to the hex string id of each configured BMC.

device_id The device to issue LED control command to on the specified board. Hexadecimal string representation of 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE is `led` - else, a 500 error is returned.

led_state (optional) `on` : Turn on the chassis identify LED. `off` : Turn off the chassis identify LED.

Request Example

```
http://opendcre:5000/opendcre/1.2/led/00000001/0005
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-led-control",
  "title": "OpenDCRE LED Control",
  "type": "object",
  "properties": {
    "led_state": {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "led_state": "on"
}
```

Errors

If a LED control action fails, or an invalid board/device combination are specified, an error (500) is returned.

1.4.9 Fan Speed

Description

The fan control command is used to get and set the fan speed in RPM for a given fan. `fan_speed` devices known to OpenDCRE that are not IPMI devices allow fan speed to be set and retrieved, while IPMI `fan_speed` devices are read-only.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/fan/<board_id>/<device_id>[/<speed_rpm>]
```

Parameters

board_id Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper byte of `board_id` reserved for future use in OpenDCRE. IPMI Bridge board has a special `board_id` of 40NNNNNN, where NNNNNN corresponds to the hex string id of each configured BMC.

device_id The device to issue fan control command to on the specified board. Hexadecimal string representation of 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE is `fan_speed` - else, a 500 error is returned.

speed_rpm (optional) Numeric decimal value to set fan speed to, in range of 0-10000.

- If `speed_rpm` is not specified, the `fan` command makes no changes, and simply retrieves and returns the fan speed in RPM. If `speed_rpm` is specified and valid, the `fan` command will return the updated fan speed value, as provided by the remote device.

Request Example

```
http://opendcre:5000/opendcre/1.2/fan/00000001/0002
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-fan-speed",
  "title": "OpenDCRE Fan Speed",
  "type": "object",
  "properties": {
    "speed_rpm": {
      "type": "number"
    }
  }
}
```

Example Response

```
{
  "speed_rpm": 4100
}
```

Errors

If a fan speed action fails, or an invalid board/device combination are specified, an error (500) is returned.

1.4.10 Test

Description

The test command may be used to verify that the OpenDCRE endpoint is up and running, but without attempting to address the device bus. The command takes no arguments, and if successful, returns a simple status message of "ok".

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/test
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.2/opendcre-1.2-test-status",
  "title": "OpenDCRE Test Status",
  "type": "object",
  "properties": {
    "status": {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "status": "ok"
}
```

Errors

If the endpoint is not running no response will be returned, as the command will always return the response above while the endpoint is functional.

1.5 OpenDCRE / OpenMistOS Release Notes

1.5.1 OpenDCRE v1.2.0 Release Notes

March 5, 2016

The OpenDCRE v1.2.0 release is a significant release, adding a large set of new features to both OpenDCRE and OpenMistOS.

This release provides a long-awaited major update to the original OpenDCRE v1.x codebase, and has excellent feature, test, and usability enhancements.

Included in this release:

- **OpenMistOS v1.1.0**
  - Kernel updated to Linux 4.1.17 for armv7l
  - Debian `jessie` (includes `systemd` support) - upgraded from `wheezy` in prior releases
  - Docker v1.10.1 and Docker Compose version v1.6.2
- **OpenDCRE v1.2.0**
  - IPMI 2.0 support added to IPMI bridge
  - **Added support via PLC and IPMI for:**
    - *fan* command (fan control)
    - *led* command (chassis "identify" LED control)
    - *location* command (physical and intra-chassis location)
    - *asset* command (asset information)
    - *boot_target* command (boot target selection)
    - *temperature*, *humidity*, *fan_speed* sensor support added
  - Normalization of OpenDCRE API command layout for consistency and future functionality
  - PLC communications (devicebus_interfaces) for v1 RPI HAT finalized
  - Emulator enhancements for new functionality
  - Improvements to code organization, PEP8 compliance, documentation
  - Testing via docker-compose (supported on RPI and Linux/MacOS)

Contributors to this release:

- **Andrew Cencini, Vapor IO (maintainer)**
  - IPMI, PLC, code, test enhancements
- **Klemente Gilbert-Espada, Vapor IO (docs, contributor)**
  - RPI `pyserial` bugfix, Sphinx documentation
- **Erick Daniszewski, Vapor IO (contributor)**
  - Test enhancements, bugfixes

Special thanks to early adopters and testers of OpenDCRE.

- OpendDCRE on GitHub.