CnuasBMC, Datasheet

Rack and sled management controllers, ORV3 power shelf firmware and Renode satellites

Document DS-CNU-031 • **Revision** C • **Issued** 20 August 2026 • **Status** Published

Item	Value
Part	`cnuas-rackmond`, `cnuas-rackmon`, ORV3 shelf firmware, `meta-cnuas`
Type	Management controller stack
Version	`9fbdddf-dirty`
Repo	PacketFive/cnuas

1. Overview

The Cnuas management stack emulates the parts of a rack that sit below the operating system: the Open Rack V3 power shelf and its satellite microcontrollers, the rack management controller that polls them, and the sled management controller that powers blades on and off.

Microcontrollers run in **Renode** rather than QEMU, because the shelf is wired with a multi-drop **RS-485** segment and Renode can model that segment directly. Real OpenBMC images run on the resulting machines, and standard tooling reaches them: `ipmitool` over the network, Redfish over HTTPS, and serial over local area network for console access.

Key features

- Open Rack V3 power shelf emulated at the firmware level, not stubbed.
- Multi-drop RS-485 segment shared by the shelf and its satellites, bridged to a TCP endpoint.
- Modbus remote terminal unit polling by the rack management controller.
- Sensor publication onto D-Bus for OpenBMC consumers, and onward as Redfish sensors.
- Redfish 1.17.0 served by `bmcweb`, with the telemetry and event services.
- Yocto layer with machine configurations for the rack, sled and switch controllers.
- Sled controller drives an emulated blade through general purpose input and output over the emulator monitor.

2. Components

Component	Role	State
ORV3 shelf firmware	Power shelf and battery backup behaviour	Builds, host tests run
`cnuas-rackmond`	Rack management daemon, polls the shelf	Builds for host and for the controller
`cnuas-rackmon`	Operator client for the daemon	Builds
`meta-cnuas`	Yocto layer, machine configurations and recipes	Layer content present
Renode platforms	Shelf and satellite machine definitions	Shelf and battery machines run, several satellites are skeletons
Sled bridge	Management controller to blade power path	Ships inside `cnuas-tools`

Component	Role	State
Front-panel firmware	Cortex-M LED and button controller over UART	Builds, host tests and Renode protocol check pass

3. Renode machine definitions

Machine	Core	Peripherals
ORV3 satellite	Cortex-M3	RS-485 through a 16550 compatible port at `0x40000000`, strap block at `0x30000000`
Battery management	Cortex-M4	RS-485 through a serial port at `0x40004400`
Power supply	Cortex-M0	Power management bus over an inter-integrated circuit slave
Fan controller	Cortex-M0	Inter-integrated circuit slave
Sequencer	Cortex-M0	Inter-integrated circuit slave with a fault line
Front panel	Cortex-M0	16550-compatible UART at `0x40004400`
Optics	Cortex-M0	Inter-integrated circuit slave, superseded

The RS-485 segment itself is a Renode plugin that joins the machines onto one multi-drop bus and exposes it on a TCP endpoint so that host software can attach.

Not every satellite carries firmware

The fan and sequencer machines are structural descriptions without firmware. The I2C optics description is superseded by the QEMU CMIS device. The ORV3 power-shelf nodes and UART front panel run real firmware.

3.1 Front-panel protocol

The front-panel MCU owns power, identify and fault LEDs and reports the live and edge-latched state of the power, identify and service buttons. Frames begin with `0xA5`, carry a protocol version, sequence number, command, payload length and CRC-8. Commands ping the firmware, read status, update masked LED bits and inject virtual button transitions. The injection command is a simulation surface; physical firmware can feed the same state machine from GPIO sampling.

4. Rack management controller

Item	Value
Daemon	`cnuas-rackmond`
Client	`cnuas-rackmon`
Shelf protocol	Modbus remote terminal unit
Shelf transport	Serial device, or a TCP endpoint on port 3485
Operator protocol	JSON over a UNIX domain socket
Operator commands	`status`, `sweep`, `ping`
Publication	D-Bus sensor objects for OpenBMC
Sensor objects	56, plus chassis inventory
Builds	Host binaries, and static binaries for the controller

The same shelf readings drive the facility twin live animation.

5. Sled management controller

Item	Value
Bridge	`cnuas-tools sledbmc bridge`
Blade control lines	Power output, reset output, power good, power on self test
Transport to the blade	Emulator monitor protocol
Redfish	HTTPS, port 2444, see section 6
Intelligent platform management interface	UDP port 623, reachable with `ipmitool -I lanplus`
Serial over local area network	Blade console through the emulator serial socket

6. Redfish service

The Redfish service is `bmcweb` from OpenBMC. Nothing in it was written for Cnuas, which is what makes a client tested against it representative. Cnuas itself is a Redfish client and serves no Redfish of its own.

Item	Value
Implementation	`bmcweb`, from the OpenBMC image
Protocol version	Redfish 1.17.0
Service root schema	`ServiceRoot.v1_15_0`
Transport	HTTPS with a self-signed certificate
Port, rack management controller	2443
Port, sled management controller	2444
Authentication	HTTP basic, and sessions under `SessionService`
Session timeout	1800 s

6.1 Resources served

Resource	Schema	Carries
`/redfish/v1`	`ServiceRoot.v1_15_0`	Version, protocol features, links
`/redfish/v1/Chassis/chassis`	`Chassis.v1_22_0`	Rack chassis, `Chassis.Reset`
`/redfish/v1/Chassis/chassis/Sensors`	`SensorCollection`	56 shelf sensors
`/redfish/v1/Systems/system`	`ComputerSystem.v1_22_0`	Power state, boot, `ComputerSystem.Reset`
`/redfish/v1/Managers/bmc`	`Manager.v1_15_0`	The controller itself
`/redfish/v1/TelemetryService`	`TelemetryService.v1_2_1`	Metric reports and triggers
`/redfish/v1/EventService`	`EventService.v1_5_0`	Events and server sent events
`/redfish/v1/AccountService`	Account collection	Local users and roles
`/redfish/v1/UpdateService`	Update service	Firmware inventory and upload
`/redfish/v1/Registries`, `/redfish/v1/JsonSchemas`	Registry collections	Message registries and schema

6.2 Sensors

The 56 D-Bus sensor objects the rack monitor publishes appear as a `SensorCollection` on the chassis. Each reading is a `Sensor.v1_11_1` with a reading, its units and its range.

Reading type	Count	Source
Power	14	Input and output of each supply, and the rack totals
Temperature	12	Supply inlets and battery pack cells
Voltage	12	Supply output rails and battery pack terminals
Current	6	Supply output
Fan speed	6	Supply fan tachometers
Utilisation	6	Battery pack state of charge

The shelf is presented through sensors rather than through `PowerSubsystem/PowerSupplies`, so that collection is empty. A power supply resource carries an inventory identity that an ORv3 unit does not report over Modbus, and populating it would mean inventing one.

6.3 Actions and query support

`ComputerSystem.Reset` accepts `On`, `ForceOn`, `ForceOff`, `PowerCycle`, `GracefulShutdown`, `GracefulRestart`, `ForceRestart` and `Nmi`, and the list is served at `Systems/system/ResetActionInfo`. `Chassis.Reset` is served alongside it.

Query feature	Supported
`$select`	Yes
`only`	Yes
`$expand`	No
`$filter`	No
Excerpt query	No
Deep `PATCH` and deep `POST`	No

The telemetry service holds up to 10 metric reports, collects no faster than one second, and offers maximum, minimum, average and summation. The event service delivers `Event` and `MetricReport` payloads and retries three times.

6.4 Reaching it

```
curl -sk -u root:0penBmc https://127.0.0.1:2443/redfish/v1/
curl -sk -u root:0penBmc \
  https://127.0.0.1:2443/redfish/v1/Chassis/chassis/Sensors/power_psu0_output_power
```

7. Yocto layer

`meta-cnuas` carries the layer configuration, machine configurations for the rack, sled and switch controllers, template samples, and recipes covering branding, the rack monitor, kernel device trees, entity manager configuration, console configuration, a package group, and host power control.

8. Validation

Area	How it is exercised
Shelf firmware	Host test target in the firmware makefile
Rack management controller	Host test target in the controller makefile
ORV3 shelf firmware	177 host checks

Area	How it is exercised
Rack management controller	45 host checks
Front-panel firmware	98 host checks plus a request and response through the Renode UART
Sled bridge and emulator monitor protocol	37 cases in `tools/tests/sledbmc/`
Control-plane BMC adapters and resources	52 pytest cases with the `test` extra

The firmware and controller targets are Makefile driven rather than part of the Python suite. Their exact commands and counts are recorded in the Validation Matrix.

9. Operating notes

- The run scripts expect externally supplied QEMU and OpenBMC images and check for them before starting.
- Renode is a separate dependency from QEMU and is needed only for the management stack.

10. Integration information

Item	Value
Repo	`PacketFive/cnuas`
Source	`bmc/firmware/orv3/`, `bmc/firmware/frontpanel/`, `bmc/rmc/`, `bmc/renode/`, `bmc/meta-cnuas/`
Bridge source	`tools/src/cnuas_tools/sledbmc/`
Emulators	Renode for the controllers, QEMU for the blade
Language	C for firmware and the controller, Python for the bridge
Related	Roadmap Epic 7

11. Revision history

Revision	Notes
A	First publication. Machine definitions, protocols and build targets read from the platform files, makefiles and sources.
B	Redfish service section added, read from a running rack management controller.
C	Reconciled implemented OpenBMC paths and validation counts; added the real UART front-panel firmware and protocol.