Devices

Robosoft

User guide

v1.1.1-r1808

Notice

8devices reserves the right to change specifications without prior notice. While the information in this manual has been compiled with great care, it may not be deemed an assurance of product characteristics. 8devices shall be liable only to the degree specified in the terms of sale and delivery. The reproduction and distribution of the documentation and software supplied with this product and the use of its contents are subject to written authorization from 8devices.

Trademarks

8devices logo is a trademark of 8devices. All other registered and unregistered trademarks in this document are the sole property of their respective owners.

Table Of Contents

1. Supported features

System Core:

- Linux 6.6 kernel
- ath11k driver for QCN9074 radio
- ath10k driver for WCN3980 radio
- GPIO fan thermal control
- Systemd system manager and init
- Dual-boot software update
- UART to network forwarding
- Multi-hardware revision support (unified image)

Radio Capabilities:

- WCN3980 radio: Dual-band 2GHz + 5GHz support
- QCN9074 radio: Configurable 2GHz or 5GHz operation
- Narrow channels control
- Frequency shift control
- Spectrum analyzer with seamelss background scan
- CCA threshold control
- MCS/NSS rate control
- WMM Tx burst for monitor interfaces

Wi-Fi Operating Modes:

- BSS Access Point (AP) mode
- BSS Station (STA) mode
- Non-associated Wi-Fi (NAW) ACKed mode
- WiFi Broadcast (WFB) mode

Management & Monitoring:

- Single YAML configuration file
- Rapid reconfiguration support
- Troubleshooting log support
- Web GUI management
- SSH service
- USB ADB access
- DHCP client with static IP fallback
- Software update with progress display
- Board and radio identification tool
- Micro-frontend web application architecture

2. Software releases

2.1 v1.1.1

Features:

- TobuFi-DVK rev5 board support
- Robonode rev2 USB2 DRD/OTG support with physical mode switch

Improvements:

- Switched to ethernet PHY interrupt-driven link detection
- Updated radio firmware premium features compatibility

Bug Fixes:

- Fixed board identifiers for proper multi-board images DTS selection
- Fixed USB VBUS GPIO polarity and pinctrl configuration for TobuFi-DVK and Robonode
- Fixed USB3 host controller regression causing stale device enumeration on Robonode

2.2 v1.1.0

Features:

- Background (agile) spectral scan
- 802.11ax (HE) raw injection support
- MCS and NSS radio rate control for Pine Radio
- Clear Channel Assessment (CCA) control for Pine Radio
- Active monitor mode support for Pine Radio
- WiFi Broadcast (WFB) stream management with TX/RX modes
- Transmit bursting support for WFB mode
- UART to network forwarding / access support
- Multi-hardware revision support (single image for multiple board variants)
- Radio firmware diagnostic logging support
- Board and radio identification tool
- GPIO fan control based on device temperature with hysteresis support
- Dynamic CPU frequency scaling and device cooling

Improvements:

- Improved spectral scan accuracy and hardware feature detection
- Spectral scan fullscreen mode and markers
- DHCP client fallback to configured static IP
- Software update with progress display
- Micro-frontend (MFE) web application architecture
- External web application support

Bug Fixes:

- Fixed narrow channels Wi-Fi setup
- Fixed DHCP client IP address not properly released when switching network mode
- Fixed spectral scan frequency axis labeling
- Fixed spectral scan support in narrow channels
- Fixed web interface connection status handling
- Fixed web interface NAW mode controls
- Fixed firmware upload running out of memory on large files
- Fixed monitor mode stability when running alongside Wi-Fi service
- WCN3980 radio country and band control now works

Known Issues:

- UART baud rate limited to 115200
- QCN9074 limited 160MHz fixed channel support
- QCN9074 limited operation in 2GHz 12/13 channels
- Occasional ADB link hang/freeze
- No DHCP server network mode support

2.3 v1.0.0

Features:

- Systemd system manager and init
- Dual-boot software update mechanism
- Narrow channels control and frequency shift control
- Spectrum analyzer
- Wi-Fi operation modes: AP, STA, NAW
- YAML configuration file and system set up
- Web GUI management interface
- Troubleshooting log support
- USB ADB and SSH system access

Known Issues:

- Monitor stability issue when using radio sniffer in parallel to Wi-Fi service.
- UART baud rate limited to 115200.
- GPIO fan does not support temperature hysteresis.
- QCN9074 limited 160MHz fixed channel support.
- QCN9074 limited operation in 2GHz 12/13 channels.
- WCN3980 radio country and band control does not work.
- Occasional ADB link hang/freeze
- Spectral scan in narrow channels not supported
- Spectral scan incorrectly reports shifted frequencies
- No DHCP server network mode support
- No HDCP client network mode fallback to static IP support

3. Robosoft Software Introduction

3.1 Overview

Robosoft is an embedded Linux software platform designed for robotic and unmanned aerial vehicle (UAV) systems. The platform provides integrated device management, network configuration, wireless connectivity, and remote operation capabilities.

3.2 Main Features

3.2.1 Network Management

The system provides flexible network configuration through network zones. Network zones are logical network segments that can include multiple interfaces (ethernet, WiFi). The system automatically creates bridges when multiple interfaces belong to the same zone, enabling seamless Layer-2 connectivity.

Network zone features: - Multiple independent network segments - Automatic bridge creation for multi-interface zones - Support for static IP and DHCP client modes - DHCP client fallback to configured static IP - Interface inheritance of zone network settings

3.2.2 WiFi Connectivity

The platform supports comprehensive WiFi capabilities across multiple radios using hostapd (Access Point) and wpa_supplicant (Station mode) services:

WiFi modes:

- **Access Point (AP)** - Broadcast WiFi network for client connections
- **Station (STA)** - Connect to existing WiFi networks
- **Mesh (NAW)** - Wireless mesh networking for extended coverage
- **WiFi Broadcast (WFB)** - Unidirectional low-latency data streaming

WiFi features:

- Dual radio support (Pine Radio + Internal Radio)
- Multiple simultaneous networks (VAPs)
- Extended frequency ranges (2.3-6.1 GHz on Pine Radio)
- WPA2/WPA3 security
- Spectrum analysis capabilities with background spectral sweep
- Advanced radio controls (channel, bandwidth, TX power, MCS, NSS)
- CCA threshold control for Pine Radio
- WiFi broadcast (WFB) mode with FEC redundancy

3.2.3 Configuration System

Configuration is managed through RoboConf, a YAML-based system that provides:

- Schema-validated configuration to prevent errors
- Automatic completion of missing parameters from device defaults
- Configuration validation before application
- Support for custom application-specific settings
- Multiple configuration methods (file editing, Python API, web interface)

3.2.4 Software Updates

The system uses SWUpdate framework with A/B partition redundancy:

- Atomic updates (complete or rollback, no partial states)
- Automatic failover if update fails to boot
- Hardware compatibility verification
- No interruption to running system during update installation
- Command line update visual progress display

3.2.5 System Monitoring

Comprehensive monitoring capabilities are available:

- Real-time WiFi statistics (signal strength, link quality, traffic)
- System resource monitoring (CPU, memory, storage, temperature)
- Service status monitoring
- System logging with journald
- Built-in diagnostic tools
- Background spectral sweep (agile scan)
- Board and radio identification (`boardinfo`)

3.2.6 User Interfaces

Multiple interfaces are available for system interaction:

- **Web Interface** - Browser-based management console (served by nginx) for configuration and monitoring
- **Command Line** - SSH or serial console access for advanced configuration
- **Python API** - Programmatic access to configuration and board information
- **REST API** - HTTP-based API for remote management (if available)

3.2.7 Serial Port Access

The system provides UART to network forwarding for serial port access. Serial ports can be accessed over the network using TCP or UDP connections, with configurable baud rate and frame format. UDP streaming mode is available for continuous data forwarding.

3.2.8 Service Management

The system uses systemd for service initialization, management, and monitoring. Services can be started, stopped, restarted, and monitored through systemd commands.

3.3 System Architecture Concepts

3.3.1 Read-Only Root Filesystem

The root filesystem is mounted read-only with a writable overlay on `/etc` for configuration changes. This design provides:

- Protection against filesystem corruption
- Consistent base system state
- Ability to reset configuration without reinstalling system
- Enhanced reliability for embedded deployments

3.3.2 A/B Partition Scheme

The system maintains two complete boot environments (Slot A and Slot B). During updates, the inactive slot is updated while the system continues running from the active slot. After successful update, the system switches to the updated slot on next boot. If the new slot fails to boot, automatic rollback to the previous slot occurs after 7 consecutive boot failures.

3.3.3 Configuration Hierarchy

Configuration follows a hierarchical validation and merging process:

- User configuration is validated against schema
- Missing parameters are filled from device defaults
- Hardware constraints from board file are enforced
- Resulting complete configuration is applied to services

3.4 Access Methods

Multiple methods are available for accessing the device:

Serial Console - Direct UART connection for low-level access, debugging, and recovery. Always available regardless of network configuration.

SSH - Secure shell access over network (WiFi, Ethernet, or USB RNDIS) for command-line management.

ADB - Android Debug Bridge over USB for file transfer and shell access without network configuration.

Web Interface - Browser-based graphical interface for configuration and monitoring.

USB Ethernet (RNDIS) - Network over USB cable for accessing device without WiFi or Ethernet configuration.

See: System Access for detailed access procedures.

3.5 Distribution Variants

The software is available in different distributions optimized for different use cases:

8dev-basic - Essential features including networking, SSH access, and software updates.

8dev-drone (Robosoft) - Complete feature set including web interface, advanced monitoring, spectrum analysis, and mesh networking support.

3.6 Version Information

The system tracks comprehensive version information:

- Software version string (e.g., "1.0.0")
- Active boot partition (A or B)
- Image name and distribution
- Hardware platform (machine type)
- Build metadata and Git commit hashes

Version information is accessible via `swinfo` command or from system files in `/etc/version` and `/lib/release/`.

3.7 Configuration File Locations

User Configuration:

- `/etc/roboconf/config.yaml` - Active system configuration

Read-Only System Files:

- `/etc/roboconf/board.yaml` - Hardware capabilities and specifications
- `/usr/share/roboconf/schema.yaml` - Configuration validation schema
- `/usr/share/roboconf/default.yaml` - Factory default configuration

3.8 Supported Hardware

The software supports multiple hardware platforms with device-specific capabilities defined in board files:

- Robonode - Production platform for unmanned systems
- TobuFi DVK - Development kit for prototyping

Each platform has specific radio capabilities, interface counts, and frequency ranges documented in its board file.

3.9 Next Steps

- System Startup - System boot process and indicators
- System Controls - System controls and software update
- Configuration - Configuration parameters description
- Monitoring - System monitoring and diagnostics

4. TobuFi-DVK Board

4.1 Overview

TobuFi-DVK is a development kit for the TobuFi System-on-Module featuring Qualcomm QCS405 SoC. This document describes board-level hardware integration and peripherals. Supported board revisions: rev3, rev4, rev5.

4.2 Peripherals

- WiFi Radio #0 (HE/11ax)
- WiFi Radio #1 (VHT/11ac)
- Ethernet port (100M/1000M)
- USB Port #1 (USB 2.0 gadget)
- USB Port #2 (USB 3.0 DRD/OTG)
- I2C EEPROM (16KB)
- 4x BLSP UART/I2C
- SD card slot
- Coax audio in/out
- Optical audio out
- HDMI port
- MIPI display
- Power LED (fixed)
- Reset button (fixed)
- Boot configuration switches (DIP, rev3/rev4) / EDL button (rev5)

4.3 Board Revisions

Feature	Rev3	Rev4	Rev5
HDMI connection	Via EP92A6 bridge	Direct SoC	Direct SoC
BLSP0 UART (block A0)	Not exported	Available	Available
Power-on behavior	Button press	Button press	Auto-start
EDL mode entry	DIP switch #1	DIP switch #1	Dedicated push button

4.4 Board Layout

4.4.1 Top View (Rev3/Rev4)

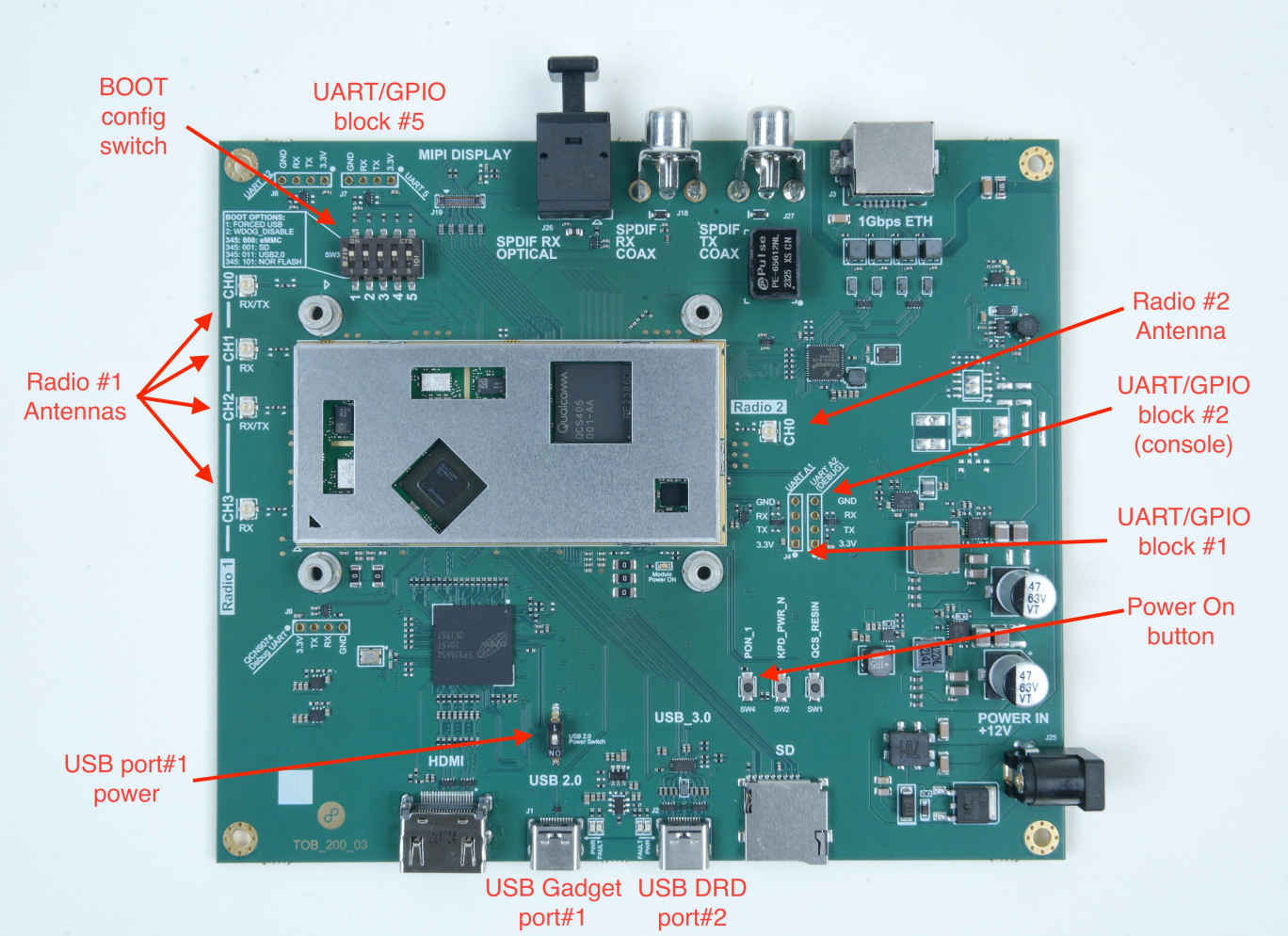


Figure 1: TobuFi-DVK rev3/rev4 board top view components

4.4.2 Bottom View (Rev3/Rev4)

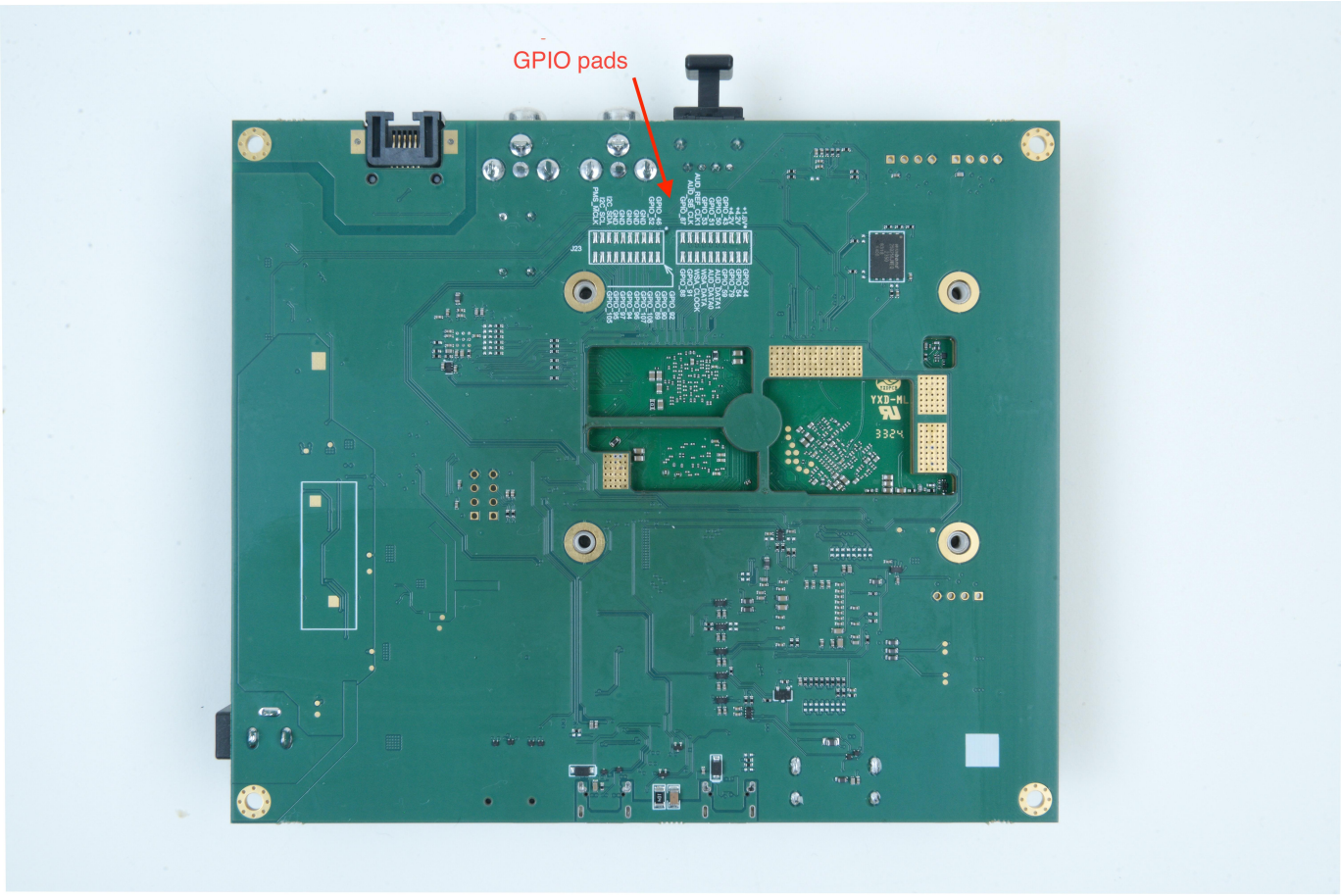


Figure 2: TobuFi-DVK rev3/rev4 board bottom view components

4.4.3 Top View (Rev5)

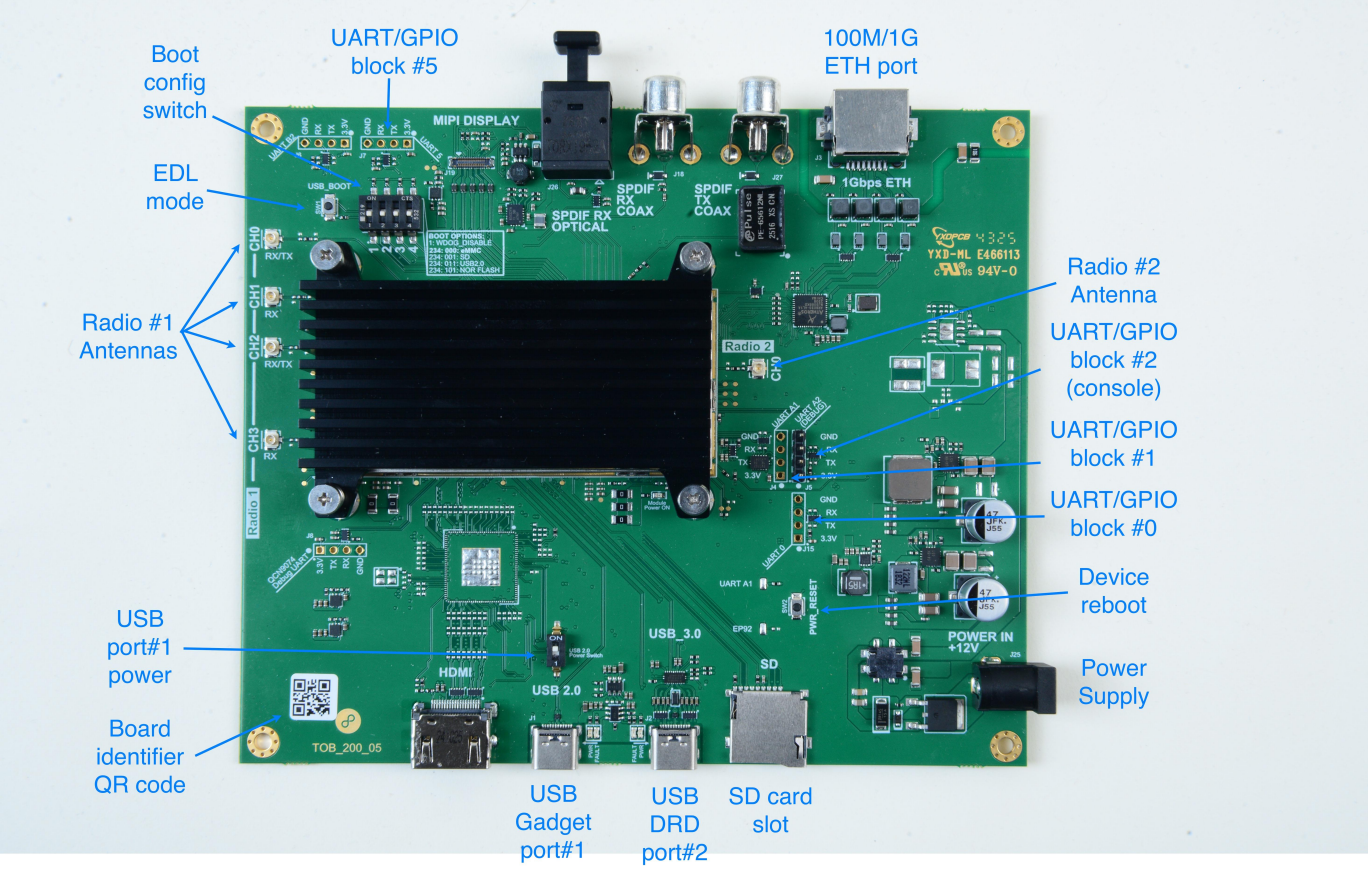


Figure 3: TobuFi-DVK rev5 board top view components

4.4.4 Bottom View (Rev5)

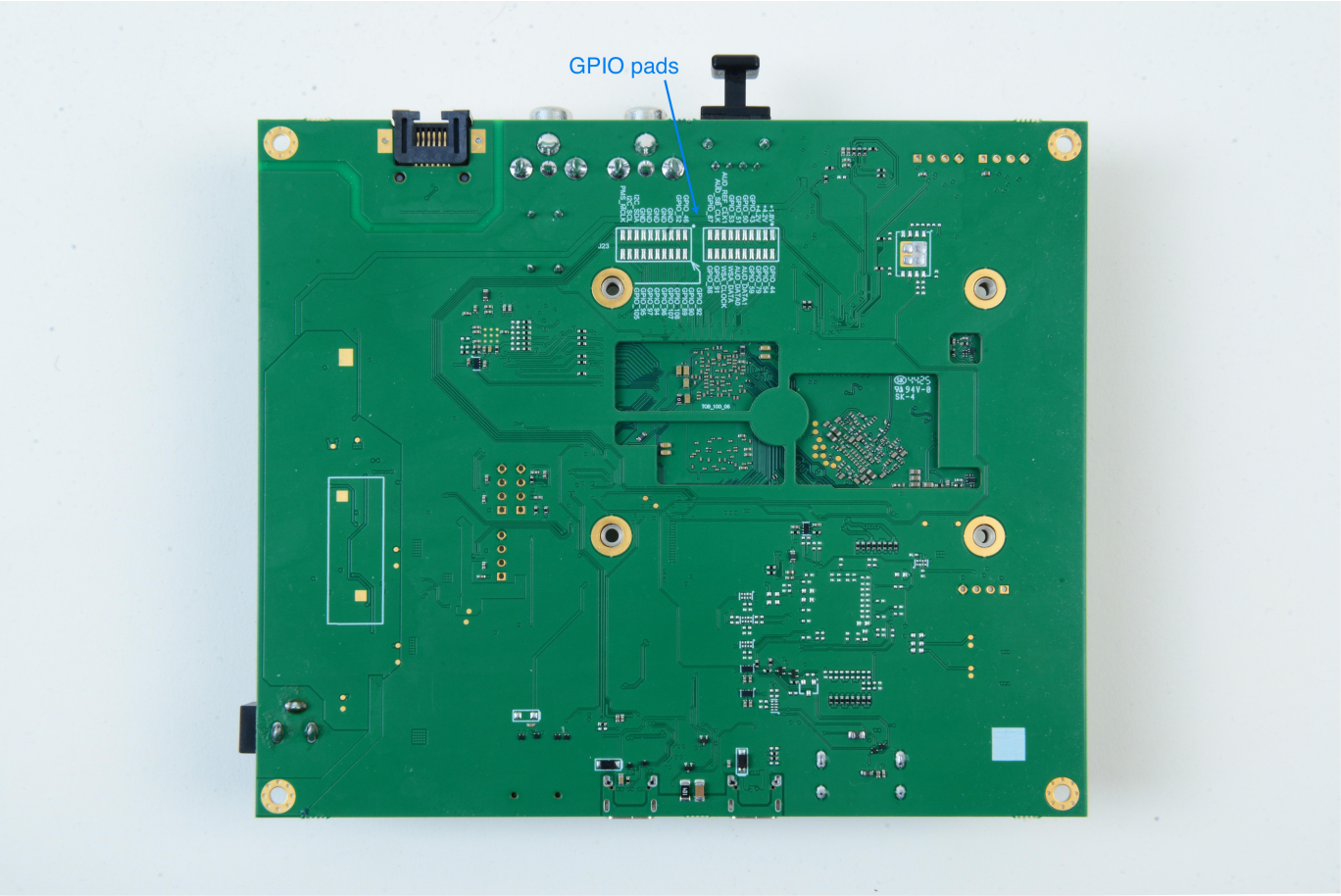


Figure 4: TobuFi-DVK rev5 board bottom view components

4.4.5 Connector Identification

Top Side:

- **Ethernet Port**: RJ45 connector
- **USB Port #1**: USB-C connector (USB 2.0 gadget)
- **USB Port #2**: USB-C connector (USB 3.0 DRD/OTG)
- **WiFi Antennas**: 4x U.FL (Radio #0), 1x U.FL (Radio #1)
- **Power Connector**: DC barrel jack (12V)
- **HDMI Port**: HDMI connector
- **SD Card Slot**: MicroSD connector
- **Audio**: Coax in/out, optical out

Bottom Side:

- **BLSP0 Header**: Pads for UART (GPIOs 30-31, rev4+ only)
- **BLSP1 Header**: Pads for UART/I2C (GPIOs 22-25)
- **BLSP2 Header**: Serial console UART (GPIOs 17-18)
- **BLSP5 Header**: Pads for UART (GPIOs 26-27)
- **Power LED**: Fixed power indicator
- **Reset Button**: System reset
- **Boot Config**: DIP switches (rev3/rev4) or EDL button (rev5)

4.5 Board Interfaces

4.5.1 I2C EEPROM

Chip: AT24C128C **Capacity:** 16KB **Interface:** BLSP1 I2C (GPIO 24-25)

Non-volatile storage for board identification and production storage. Accessible through standard Linux I2C interface.

4.5.2 WiFi Radio #0 (Pine)

Parameter	Description
Radio Chip	PCIe/QCN9074
WiFi Standard	IEEE 802.11ax (WiFi 6/6E)
Channel Width	20/40/80/160 MHz
Operation Band	2GHz + 5GHz or 2GHz + 6GHz
Operation Mode	2x4 (2T4R)
Antenna Connectors	4x U.FL
Max Conducted 2GHz Tx Power (aggregate)	32 dBm
Max Conducted 5GHz Tx power (aggregate)	29 dBm
Max Conducted 6GHz Tx Power (aggregate)	29 dBm
2GHz Frequency Range	2360–3150 MHz
5GHz Frequency Range	4550–6630 MHz
6GHz Frequency Range	5325–7495 MHz

High-performance WiFi radio supporting WiFi 6E (802.11ax) with MIMO capability. Suitable for high-throughput wireless applications, mesh networking, and dual-band simultaneous operation.

4.5.3 WiFi Radio #1

Parameter	Description
Radio Chip	SNOC/WCN3980
WiFi Standard	IEEE 802.11ac (WiFi 5)
Channel Width	20/40/80 MHz
Operation Band	2GHz + 5GHz
Operation Mode	1x1 (1T1R)
Antenna Connectors	1x U.FL
Max 2GHz Tx Power	16 dBm
Max 5GHz Tx power	16 dBm
2.4 GHz Frequency Range	2412-2484 MHz
5 GHz Frequency Range	5180-5825 MHz

Integrated WiFi radio supporting dual-band operation. Suitable for management interface, station mode connectivity, or additional access point deployment.

4.5.4 Ethernet Port

Parameter	Description
Transceiver Chip	RGMII/AR8033A
Speed	10/100/1000 Mbps
Duplex	Half/Full
MDI/MDI-X	Auto-crossover

Gigabit Ethernet port with RJ45 connector. Supports network connectivity for wired applications, device management, and high-bandwidth data transfer.

4.5.5 USB Port #1

Parameter	Description
Standard	USB 2.0
Speed	480 Mbps (HS)
Connector	USB-C
Mode	Gadget only
Identifier	usb0

Gadget port for host PC access. ADB debugging interface is available through this port.

4.5.6 USB Port #2

Parameter	Description
Standard	USB 3.0
Speed	5 Gbps (SS) / 480 Mbps (HS)
Connector	USB-C
Mode	DRD (Host/Gadget/OTG)
Switching	Automatic electrical
Identifier	usb1

USB dual-role (DRD) on-the-go (OTG) port with automatic runtime switching between host and device modes. In host mode, supports connecting USB flash drives, USB cameras, and other standard USB peripherals like mice and keyboards. In device mode, functions as a USB gadget port that can be configured with various functions in software.

4.5.7 Boot Configuration (Rev3/Rev4)

Boot configuration DIP switches allow switching the boot devices order. The following combinations are supported:

Mode	SW #1	SW #2	SW #3	SW #4	SW #5
EDL recovery	ON	-	-	-	-
eMMC → SD → USB2	OFF	-	OFF	OFF	OFF
SD → eMMC → EDL	OFF	-	ON	OFF	OFF
eMMC → EDL	OFF	-	OFF	ON	OFF
USB2	OFF	-	ON	ON	OFF
NAND → EDL	OFF	-	OFF	OFF	ON
NOR → EDL	OFF	-	ON	OFF	ON

Dashes (-) indicate that switch position is not relevant.

4.5.8 Boot Configuration (Rev5)

Rev5 uses a dedicated EDL mode push button instead of DIP switches. The EDL entry procedure is the same as Robonode: hold the EDL button pressed when power cycling or rebooting the device.

4.5.9 Power-On and Reset

Rev3/Rev4: Device requires pressing the "Power ON" button after applying power. The PON button is used to start the device on every power cycle. Button press is not required when performing a normal Linux reboot.

Rev5: Device starts automatically when power is applied. The PON button is repurposed as a hard device reset.

4.6 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi SoM - TobuFi SoM (System-on-Module) details
- TobuFi-DVK GPIO - Board GPIO interfaces and control
- TobuFi-DVK BLSP - Board UART and I2C interfaces
- TobuFi-DVK Setup - Initial board setup and flashing
- TobuFi-DVK Initialisation - Board initialization sequence

5. TobuFi-DVK Setup

This guide covers initial device setup and hands-on launch procedures for the TobuFi-DVK board.

For board hardware specifications and connector locations, refer to TobuFi-DVK Hardware.

5.1 Required Equipment

- Compatible DC power supply (12V, 1.5A minimum)
- For console access: Serial console cable (3.3V TTL UART)
- For console access: Serial terminal software (minicom, screen, PuTTY, etc.)
- For device recovery: USB Type-C cable
- For device recovery: Linux PC with qdl tool installed or Windows PC with QFIL

5.2 Power Requirements

The TobuFi-DVK board requires a **12V DC** power supply capable of supplying at least **1.5A**.

Power On (Rev3/Rev4):

- Connect power supply to the barrel jack
- Press "Power ON" button to start the device

This procedure is needed every time the device is power cycled. However it is not required when performing a normal Linux reboot.

Power On (Rev5):

- Connect power supply to the barrel jack — device starts automatically

Refer to TobuFi-DVK Hardware for power connector location on the PCB.

5.3 Serial Console Access

To access the device via serial console:

- Connect 3.3V TTL serial console cable to the serial console connector
- Configure serial terminal with appropriate settings
- Follow System Access Guide for connection details

Refer to TobuFi-DVK Hardware for serial console connector location on the PCB.

5.4 USB Recovery

5.4.1 Emergency Download (EDL) Mode

EDL (Emergency Download) mode is a low-level boot mode in the Qualcomm SoC that enables device recovery and initial programming by exposing a USB interface for flash programming.

Use cases:

- Initial factory programming of blank devices
- Recovery from bootloader corruption
- Full device reflashing
- Partition table restoration

Device appears as USB `05c6:9008` (Qualcomm HS-USB QDLoader 9008) in EDL mode.

Important: EDL recovery installs legacy software version v0.x which can subsequently be upgraded to open source software version v1.x using USB flashing (see USB Flashing section).

Warning: EDL recovery completely erases and reprograms device flash. Use correct recovery images for TobuFi-DVK hardware to avoid permanent damage.

5.4.2 Entering EDL Mode (Rev3/Rev4)

- Connect USB gadget port#1 to PC
- Set boot config DIP switch #1 to ON
- Power cycle the device
- Press and release "Power ON" button
- Device enters EDL mode and appears as `05c6:9008`

Refer to TobuFi-DVK Hardware for button and switch locations on the PCB.

5.4.3 Exiting EDL Mode (Rev3/Rev4)

- Set DIP switch #1 to OFF
- Power cycle the device
- Press and release "Power ON" button to start normal device boot

5.4.4 Entering EDL Mode (Rev5)

- Connect USB gadget port#1 to PC

- Press and hold "EDL mode" button
- Press and release "Device reboot" button
- Release "EDL mode" button
- Device enters EDL mode and appears as `05c6:9008`

Refer to TobuFi-DVK Hardware for button locations on the PCB.

5.4.5 Obtaining Recovery Package

Recovery flash image packages are available from 8devices support. Contact 8devices support to obtain the appropriate recovery package for your device.

5.4.6 EDL Recovery Tools

EDL recovery can be performed using different tools depending on your operating system:

Linux:

- Recommended: `edl_flash.py` script (included in recovery package, Linux only)
- Alternative: QDL tool from https://github.com/linux-msm/qdl

Windows:

- QFIL (Qualcomm Flash Image Loader) from Qualcomm Product Support Tools (QPST)

5.4.7 EDL Recovery Using edl_flash.py (Linux only)

The `edl_flash.py` script provides automated EDL recovery on Linux systems.

Recovery Steps:

- Connect USB gadget port#1 to PC
- Enter EDL recovery mode (see Entering EDL Mode or Rev5)
- Execute EDL flashing script from software flash image package:

```
python3 edl_flash.py
```

- Power cycle the device to boot new firmware

Multiple EDL Devices:

When multiple devices are in EDL state, specify device by serial number:

```
# Flash specific device by serial
python3 edl_flash.py <serial>

# List available EDL devices
python3 edl_flash.py -l
```

5.4.8 EDL Recovery Using QDL (Linux)

Alternative method using QDL tool:

```
# Verify device enumeration
lsusb | grep 05c6:9008

# Flash recovery images
qdl --storage ufs prog_firehose.mbn rawprogram.xml patch.xml
```

Required images: prog_firehose.mbn (programmer), rawprogram.xml (partition layout), patch.xml (patching instructions), partition images (bootloader, kernel, rootfs).

5.4.9 EDL Recovery Using QFIL (Windows)

Use QFIL from Qualcomm Product Support Tools package. Refer to QFIL documentation for flashing procedures.

5.5 USB Flashing

Fastboot provides partition-level flashing capabilities through the bootloader interface. This method is used to upgrade devices from legacy software v0.x to open source software v1.x.

5.5.1 Prerequisites

- ADB (Android Debug Bridge) and Fastboot tools
- Windows: USB driver for Android devices

5.5.2 Fastboot Flashing Using fastboot_flash.py

The `fastboot_flash.py` script automates fastboot flashing operations. The script automatically detects device state and enters fastboot mode if needed.

Flashing Procedure:

```
# Flash boot and system partitions
python3 fastboot_flash.py --boot boot-image.img --system rootfs-image.img
```

Multiple Devices:

When multiple devices are connected, specify device by serial number:

```
# Specify device by serial number
python3 fastboot_flash.py <serial> --boot boot.img --system rootfs.img

# Check available devices
fastboot devices
```

The script will:

- Automatically detect if device is in ADB mode and reboot to fastboot
- Wait for device to appear in fastboot mode (up to 120 seconds)
- Flash specified partitions
- Automatically reboot device after successful flashing

5.6 Related Documentation

- TobuFi-DVK Hardware - Board hardware periphery briefing
- TobuFi-DVK BLSP - Board UART and I2C interfaces
- TobuFi-DVK Initialisation - Board hardware initialisation briefing

6. Robonode Board

6.1 Overview

Robonode is a development board for unmanned systems based on the TobuFi System-on-Module featuring Qualcomm QCS405 SoC. This document describes board-level hardware integration and peripherals.

6.2 Peripherals

- WiFi Radio #0 (HE/11ax)
- WiFi Radio #1 (VHT/11ac)
- Ethernet port (100M/1000M)
- USB Port #1 (USB 2.0 DRD/OTG)
- USB Port #2 (USB 3.0 host)
- I2C EEPROM (16KB)
- 4x BLSP UART/I2C
- 3x GPIO LEDs
- GPIO Push button
- GPIO Switch toggle
- Reboot button
- USB recovery button

6.3 Board Layout

6.3.1 Top View

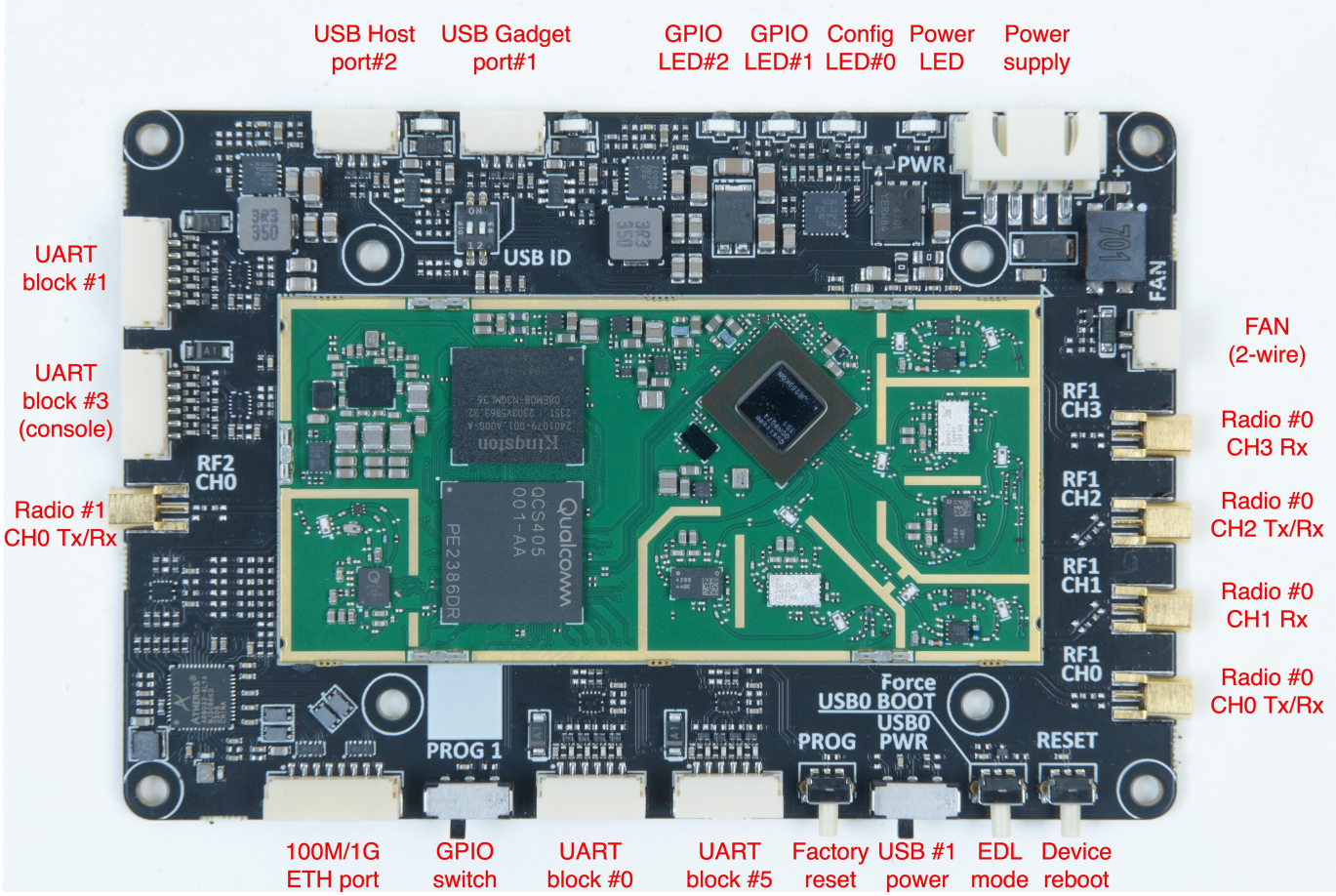


Figure 1: Robonode board top view components

6.3.2 Bottom View

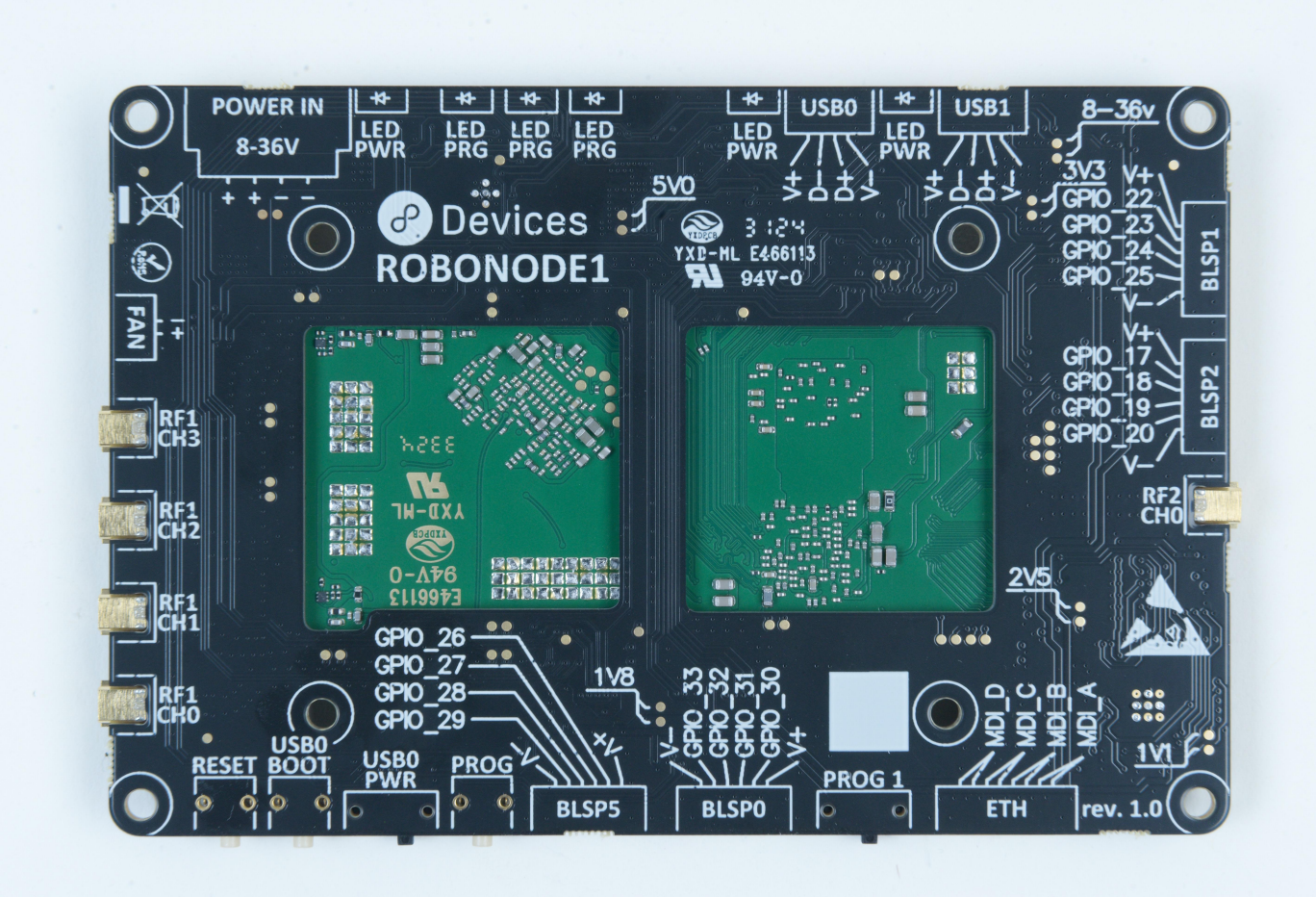


Figure 2: Robonode board bottom view components

6.3.3 Connector Identification

Top Side:

- **Ethernet Port**: RJ45 connector, labeled "ETH"
- **USB Port #1**: USB-C connector (left side)
- **USB Port #2**: USB-C connector (right side)
- **WiFi Antennas**: 4x MMCX (Radio #0), 1x MMCX (Radio #1)
- **Power Connector**: DC barrel jack (8-36V)

Bottom Side:

- **BLSP0 Header**: 4-pin header for UART/I2C (GPIOs 30-33)
- **BLSP1 Header**: 4-pin header for UART/I2C (GPIOs 22-25)
- **BLSP2 Header**: 4-pin header for UART/I2C (GPIOs 17-20), includes serial console
- **BLSP5 Header**: 4-pin header for UART/I2C (GPIOs 26-29)
- **LED0, LED1, LED2**: User-defined GPIO LEDs
- **Push Button**: User-defined GPIO button
- **Slide Switch**: User-defined GPIO switch
- **Reboot Button**: System reset button
- **Recovery Button**: EDL mode entry button

6.4 Board Interfaces

6.4.1 I2C EEPROM

Chip: AT24C128C **Capacity:** 16KB **Interface:** BLSP1 I2C (GPIO 24-25)

Non-volatile storage for board identification and production storage. Accessible through standard Linux I2C interface.

6.4.2 WiFi Radio #0 (Pine)

Parameter	Description
Radio Chip	PCIe/QCN9074
WiFi Standard	IEEE 802.11ax (WiFi 6/6E)
Channel Width	20/40/80/160 MHz
Operation Band	2GHz + 5GHz or 2GHz + 6GHz
Operation Mode	2x4 (2T4R)
Antenna Connectors	4x MMCX
Max Conducted 2GHz Tx Power (aggregate)	32 dBm
Max Conducted 5GHz Tx power (aggregate)	29 dBm
Max Conducted 6GHz Tx Power (aggregate)	29 dBm
2GHz Frequency Range	2360–3150 MHz
5GHz Frequency Range	4550–6630 MHz
6GHz Frequency Range	5325–7495 MHz

High-performance WiFi radio supporting WiFi 6E (802.11ax) with MIMO capability. Suitable for high-throughput wireless applications, mesh networking, and dual-band simultaneous operation.

6.4.3 WiFi Radio #1

Parameter	Description
Radio Chip	SNOC/WCN3980
WiFi Standard	IEEE 802.11ac (WiFi 5)
Channel Width	20/40/80 MHz
Operation Band	2GHz + 5GHz
Operation Mode	1x1 (1T1R)
Antenna Connectors	1x MMCX
Max 2GHz Tx Power	16 dBm
Max 5GHz Tx power	16 dBm
2.4 GHz Frequency Range	2412-2484 MHz
5 GHz Frequency Range	5180-5825 MHz

Integrated WiFi radio supporting dual-band operation. Suitable for management interface, station mode connectivity, or additional access point deployment.

6.4.4 Ethernet Port

Parameter	Description
Trancever Chip	RGMII/AR8033A
Speed	10/100/1000 Mbps
Duplex	Half/Full
MDI/MDI-X	Auto-crossover

Gigabit Ethernet port with RJ45 connector. Supports network connectivity for wired applications, device management, and high-bandwidth data transfer.

6.4.5 USB Port #1

Parameter	Description
Standard	USB 2.0
Speed	480 Mbps (HS)
Connector	USB-C
Mode	DRD (Host/Gadget/OTG)
Power	Switch-controlled
Identifier	usb0

Dual-role port capable of host, device and OTG modes with manual power switch control. In device mode, provides access to ADB debugging interface for development and configuration. In host mode, supports standard USB peripherals.

USB Port #1 Power Switch

Slide switch controls USB mode and power supply. When toggled on USB port is working in host controller mode, and port is powered locally, when toggled off USB port is working in device mode and it expects to be powered from the host device.

Enable USB power when desirable to use port to connect additional periphery, i.e. USB camera. Keep USB power disabled to keep port in device mode, i.e. for ADB access, USB storage or USB ethernet adapter mode.

6.4.6 USB Port #2

Parameter	Description
Standard	USB 2.0
Speed	480 Mbps (HS)
Connector	USB-C
Mode	Host only
Power	Always powered
Identifier	usb1

Host-only port for connecting USB flash drives, USB cameras, and other standard USB peripherals like mice and keyboards.

6.4.7 LED #0

GPIO: 47

Application controlled LED dedicated for system configuration state feedback. LED is fully controlled by application software therefore may be repurposed on demand, changing application source code and rebuilding software.

6.4.8 LED #1

GPIO: 48

Software controlled LED. Currently unused, dedicated for future application needs.

6.4.9 LED #2

GPIO: 52

Software controlled LED, currently unused, dedicated for future application needs.

For LED control via sysfs and device tree configuration, see GPIO Configuration.

6.4.10 GPIO Push Button

GPIO: 58

Software controlled push button. Pressing and holding the button for at least 10 seconds triggers factory reset, resetting device configuration to factory defaults. This is also indicated by the configuration LED slowly blinking while the button is being held.

6.4.11 GPIO Switch Toggle

GPIO: 51

Software controlled slide switch, currently unused, dedicated for future application needs.

For button and switch programming via Linux input subsystem, see GPIO Configuration.

6.4.12 Reboot Button

Pressing and releasing the button restarts the device.

6.4.13 USB Recovery Button

Hold the button pressed when power cycling or rebooting to enter EDL (Emergency Download) mode for software recovery or initial device flashing.

6.5 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi SoM - TobuFi SoM (System-on-Module) details
- Robonode GPIO - Board GPIO interfaces and control
- Robonode BLSP - Board UART and I2C interfaces
- Robonode Initialization - Board initialization sequence

7. Robonode Setup

This guide covers initial device setup and hands-on launch procedures for the Robonode board.

For board hardware specifications and connector locations, refer to Robonode Hardware.

7.1 Required Equipment

- Compatible DC power supply
- For console access: Serial console cable (3.3V TTL UART)
- For console access: Serial terminal software (minicom, screen, PuTTY, etc.)
- For device recovery: USB Type-C cable
- For device recovery: Linux PC with qdl tool installed or Windows PC with QFIL

7.2 Power Requirements

The Robonode board requires a DC power supply ranging from **8V to 36V** capable of supplying ~24W peak power.

Power On:

- Connect power supply to the power connector
- Device automatically starts up

Refer to Robonode Hardware for power connector location on the PCB.

7.3 Serial Console Access

To access the device via serial console:

- Connect 3.3V TTL serial console cable to the serial console connector
- Configure serial terminal with appropriate settings
- Follow System Access Guide for connection details

Refer to Robonode Hardware for serial console connector location on the PCB.

7.4 USB Recovery

7.4.1 Emergency Download (EDL) Mode

EDL (Emergency Download) mode is a low-level boot mode in the Qualcomm SoC that enables device recovery and initial programming by exposing a USB interface for flash programming.

Use cases:

- Initial factory programming of blank devices
- Recovery from bootloader corruption
- Full device reflashing
- Partition table restoration

Device appears as USB `05c6:9008` (Qualcomm HS-USB QDLoader 9008) in EDL mode.

Important: EDL recovery installs legacy software version v0.x which can subsequently be upgraded to open source software version v1.x using USB flashing (see USB Flashing section).

Warning: EDL recovery completely erases and reprograms device flash. Use correct recovery images for Robonode hardware to avoid permanent damage.

7.4.2 Entering EDL Mode

- Connect USB gadget port#1 to PC
- Press and hold "EDL mode" button
- Press and release "Device reboot" button
- Release "EDL mode" button
- Device enters EDL mode and appears as `05c6:9008`

Refer to Robonode Hardware for button locations on the PCB.

7.4.3 Obtaining Recovery Package

Recovery flash image packages are available from 8devices support. Contact 8devices support to obtain the appropriate recovery package for your device.

7.4.4 EDL Recovery Tools

EDL recovery can be performed using different tools depending on your operating system:

Linux:

- Recommended: `edl_flash.py` script (included in recovery package, Linux only)
- Alternative: QDL tool from https://github.com/linux-msm/qdl

Windows:

- QFIL (Qualcomm Flash Image Loader) from Qualcomm Product Support Tools (QPST)

7.4.5 EDL Recovery Using edl_flash.py (Linux only)

The `edl_flash.py` script provides automated EDL recovery on Linux systems.

Recovery Steps:

- Connect USB gadget port#1 to PC
- Enter EDL recovery mode (see Entering EDL Mode)
- Execute EDL flashing script from software flash image package:

```
python3 edl_flash.py
```

- Power cycle the device to boot new firmware

Multiple EDL Devices:

When multiple devices are in EDL state, specify device by serial number:

```
# Flash specific device by serial
python3 edl_flash.py <serial>

# List available EDL devices
python3 edl_flash.py -l
```

7.4.6 EDL Recovery Using QDL (Linux)

Alternative method using QDL tool:

```
# Verify device enumeration
lsusb | grep 05c6:9008

# Flash recovery images
qdl --storage ufs prog_firehose.mbn rawprogram.xml patch.xml
```

Required images: prog_firehose.mbn (programmer), rawprogram.xml (partition layout), patch.xml (patching instructions), partition images (bootloader, kernel, rootfs).

7.4.7 EDL Recovery Using QFIL (Windows)

Use QFIL from Qualcomm Product Support Tools package. Refer to QFIL documentation for flashing procedures.

7.5 USB Flashing

Fastboot provides partition-level flashing capabilities through the bootloader interface. This method is used to upgrade devices from legacy software v0.x to open source software v1.x.

7.5.1 Prerequisites

- ADB (Android Debug Bridge) and Fastboot tools
- Windows: USB driver for Android devices

7.5.2 Fastboot Flashing Using fastboot_flash.py

The `fastboot_flash.py` script automates fastboot flashing operations. The script automatically detects device state and enters fastboot mode if needed.

Flashing Procedure:

```
# Flash boot and system partitions
python3 fastboot_flash.py --boot boot-image.img --system rootfs-image.img
```

Multiple Devices:

When multiple devices are connected, specify device by serial number:

```
# Specify device by serial number
python3 fastboot_flash.py <serial> --boot boot.img --system rootfs.img

# Check available devices
fastboot devices
```

The script will:

- Automatically detect if device is in ADB mode and reboot to fastboot
- Wait for device to appear in fastboot mode (up to 120 seconds)
- Flash specified partitions
- Automatically reboot device after successful flashing

7.6 Related Documentation

- Robonode Hardware - Board hardware periphery briefing
- Robonode Initialization - Board hardware initialisation briefing
- Robonode BLSP - Board UART and I2C interfaces

8. System Startup

8.1 Overview

This section describes the system boot process, expected startup behavior, and startup indicators. Understanding the boot sequence helps verify the system is operating correctly and aids in troubleshooting boot-related issues.

8.2 Boot Sequence

8.2.1 Power-On to Login

The system boot process follows these stages:

- **Bootloader Initialization** - Hardware initialization and partition selection
- **Kernel Loading** - Linux kernel loads from active boot partition
- **Root Filesystem Mount** - Root filesystem mounted read-only with writable overlay
- **System Services Start** - Essential services initialize in dependency order
- **Network Services Start** - Network zones, bridges, and interfaces configured
- **Application Services Start** - WiFi, web interface, and monitoring services start
- **System Ready** - Login prompt available, services operational

8.2.2 Expected Boot Time

Typical boot times from power-on to system ready:

- **Initial bootloader**: 3-5 seconds
- **Kernel and essential services**: 10-15 seconds
- **Network and application services**: 10-20 seconds
- **Total boot time**: 30-45 seconds

Boot time may vary depending on:

- Number of configured network interfaces
- WiFi scanning and connection establishment (station mode)
- Storage device speed
- System configuration complexity

8.3 Boot Partition Selection

8.3.1 Active Partition

The bootloader selects the active partition based on:

- Boot partition variable (set by update process)
- Boot failure counter (tracks consecutive failed boots)
- Fallback to alternate partition after 7 failures

The currently active partition (A or B) is visible in system version information:

```
swinfo | grep "SW partition"
```

8.3.2 Automatic Failover

If the system fails to boot successfully:

- Bootloader increments boot failure counter
- After 7 consecutive failures, bootloader switches to alternate partition
- System boots from previous working partition
- Boot counter resets after successful boot

This automatic failover ensures recovery from failed updates without user intervention.

8.4 Startup Indicators

8.4.1 Serial Console Output

During boot, the serial console displays detailed boot messages:

- Bootloader messages and partition selection
- Kernel initialization and hardware detection
- Service startup messages
- Error messages if services fail to start

Serial console access requires connecting UART cable to serial port. See System Access for serial connection details.

8.4.2 Network Availability

Network interfaces become available in stages:

- **Ethernet interfaces** - Available shortly after network service starts
- **WiFi Access Points** - Available after hostapd service starts (1-5 seconds)
- **WiFi Station connections** - Available after scanning and authentication complete (5-30 seconds)
- **DHCP address assignment** - Additional time if using DHCP client mode

8.4.3 Web Interface Availability

The web management interface becomes available after:

- Network interfaces are operational
- Web server service starts
- Web application initializes

Typical web interface availability: 30-45 seconds after power-on.

Access web interface at device IP address (default: `http://192.168.2.1` or configured address).

8.5 Verifying System Startup

8.5.1 Check System Status

After boot, verify system is operational:

Check system uptime:

```
uptime
```

Check failed services:

```
systemctl --failed
```

If no failed services are listed, system startup was successful.

Check network interfaces:

```
ip addr show
```

Verify expected interfaces are present and have IP addresses assigned.

Check WiFi interfaces:

```
iw dev
```

Verify WiFi interfaces are present and in correct mode (AP, station, etc.).

8.5.2 Check Service Status

Verify key services are running:

Network manager:

```
systemctl status netman
```

WiFi services (if configured):

```
systemctl status hostapd@wlan0
systemctl status wpa_supplicant@wlan1
```

All services should show `active (running)` status.

8.6 Boot Logs

8.6.1 Viewing Boot Messages

Boot logs are accessible through journald:

View current boot logs:

```
journalctl -b
```

View previous boot logs:

```
journalctl -b -1
```

View kernel messages:

```
dmesg
```

View service startup logs:

```
journalctl -u netman
journalctl -u hostapd@wlan0
journalctl -u wpa_supplicant@wlan1
```

8.6.2 Boot Log Analysis

Boot logs help identify:

- Hardware detection issues
- Service startup failures
- Configuration errors
- Network initialization problems

Look for `[FAILED]` or `ERROR` messages in boot logs to identify issues.

8.7 Startup Configuration

8.7.1 Service Dependencies

Services start in dependency order:

- **Essential services** - systemd, filesystem mounts, logging
- **Network foundation** - Network manager, interface initialization
- **Network services** - DHCP client/server, DNS
- **Application services** - WiFi, web interface, monitoring

If a service fails, dependent services may not start.

8.7.2 Automatic Service Startup

Services configured to start automatically at boot:

- Network manager (netman)
- WiFi services (hostapd, wpa_supplicant)
- Web server (nginx)
- Monitoring services (libwifistat, spectrald)

Service autostart is controlled by systemd unit files.

8.8 First Boot Behavior

8.8.1 Initial Configuration

On first boot with factory defaults:

- Default network zones are configured
- Default IP addresses are assigned
- WiFi radios are configured per board defaults
- Web interface is accessible at default IP

Default configuration is device-specific and documented in board documentation.

8.9 Troubleshooting Boot Issues

8.9.1 System Not Booting

If system shows no activity or fails to boot:

- Check power supply and connections
- Verify power LED status
- Connect to serial console to view boot messages
- Check for bootloader error messages

If system fails repeatedly, automatic failover activates after 7 consecutive boot failures and reverts to previous partition.

8.9.2 System Boots But Services Fail

If services fail to start during boot:

- Check system logs for errors:

```
journalctl -b | grep -i error
```

- Check failed services:

```
systemctl --failed
```

- Review specific service logs:

```
journalctl -u <service>
```

- Collect diagnostic data:

```
trouble
```

8.10 Next Steps

- Controls - System controls and software update
- Configuration - Configuration parameters description
- Monitoring - System monitoring and diagnostics

9. System Access

This guide describes various methods to access and interact with the Robonode device. Multiple access methods are available depending on your connection type and use case:

- **Console access** - Direct serial connection via UART
- **SSH connection** - Secure shell over network
- **ADB access** - Android Debug Bridge via USB
- **GUI access** - Web interface over network
- **USB ethernet** - Network over USB cable (RNDIS)

9.1 Console access

Connect serial console cable to board's UART TTL header. For serial console TTL header location, refer to: - TobuFi-DVK Hardware - Robonode Hardware

Serial Port Configuration:

Default mode: 115200 8N1.

Parameter	Value
Port	/dev/ttyUSB# (Linux) or COM# (Windows)
Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

Serial Connection Wiring:

- Pin 1: GND (Black wire)
- Pin 2: RX (White wire - connects to TX on FTDI)
- Pin 3: TX (Green wire - connects to RX on FTDI)
- Pin 4: VCC (Red wire - do NOT connect)

Run desired terminal emulator:

- picocom
- microcom
- minicom
- etc.

Examples:

```
picocom -b 115200 /dev/ttyUSB0

microcom -s 115200 -p /dev/ttyUSB0

minicom -D /dev/ttyUSB0 -b 115200

screen /dev/ttyUSB0 115200

# Using PuTTY (Windows)
# Connection type: Serial
# Serial line: COMX (check Device Manager)
# Speed: 115200
```

Upon successful connection user name prompt will be displayed. Default user name is `root`, no password is required.

9.2 SSH connection

SSH connection requires connectivity over IP networking. Connecting to the device may be done either over:

- Wi-Fi
- Wired ethernet via UTP cable
- Virtual RNDIS/modem interface via USB cable

Target device and host systems IP address subnets must match. For configuration details to set up target device network, refer to Configuration Parameters for network configuration.

9.2.1 Shell access

Establish SSH connection using desired SSH client:

```
ssh root@192.168.2.1
```

Default user name is `root`, no password is required.

9.2.2 Upload and download files

File operations are supported either via legacy scp or sftp protocols thereby any scp/sftp protocol utility should work (scp, sftp, winscp, filezilla, etc.).

Upload files to device:

```
scp <local files...> root@192.168.2.1:<remote path>
```

Download files from device:

```
scp root@192.168.2.1:<remote file> <local path>
```

9.3 ADB access

ADB connection uses a USB bus configured in device mode providing ADB function and companion `adbd` service to handle host requests. In contrast, `fastboot` provides similar functionality through USB but is controlled by lower level firmware (i.e. bootloader).

To setup the USB ADB connection:

- Connect the device's USB gadget port to the PC using a USB cable capable of data transfer. Refer to TobuFi-DVK Hardware or Robonode Hardware for USB gadget port description and location on board.
- Use ADB tools on your host computer to access device as described in section below.

9.3.1 Required Software Installation

Before using ADB, you'll need to install some software on your computer. Follow the instructions for your operating system:

Linux (Ubuntu/Debian)

Open a terminal and run:

```
sudo apt install android-tools-adb android-tools-fastboot
```

MacOS

Using Homebrew package manager, open a terminal and run:

```
brew install android-platform-tools
```

Windows

- Download and install Android SDK Platform Tools from: https://developer.android.com/tools/releases/platform-tools
- Install USB drivers (if needed):
- Google USB Driver: https://developer.android.com/studio/run/win-usb
- Or Universal ADB Driver: https://adb.clockworkmod.com/

9.3.2 Device Serial Number

Each device has a unique serial number embedded in the kernel command line during boot. This serial number: - Uniquely identifies the device for ADB operations - Is required when multiple devices are connected - Appears in the output of `adb devices` command

For technical details on how the serial number is retrieved, see TobuFi-DVK Initialisation or Robonode Initialization.

9.3.3 Listing devices

```
adb devices
```

Lists all connected ADB devices with their serial numbers.

9.3.4 Shell access

```
adb [-s <serial>] shell
```

ADB shell is accessed using command above. When there is only single device connected to host PC, serial number may be omitted. When host PC detects more than one ADB device connected, serial number is required. Available devices serial numbers can be retrieved by listing the devices or directly retrieving device serial number.

9.3.5 Upload and download files

Upload files to device:

```
adb [-s <serial>] push <local files...> <remote path>
```

Download files from device:

```
adb [-s <serial>] pull <remote files...> <local path>
```

9.4 GUI access

The device hosts a web-based management interface that can be accessed through a standard web browser over the network. The web UI provides a graphical interface for device configuration, monitoring, and management without requiring command-line access.

To access the device's web management interface:

- Ensure your computer is connected to the same network as the device, either through Wi-Fi, Ethernet, or USB RNDIS connection.
- Open a web browser on your computer and navigate to the device's IP address:

```
http://192.168.2.1
```

9.5 USB ethernet

USB ethernet uses RNDIS (Remote Network Driver Interface Specification) protocol to provide network connectivity over a USB cable by creating a virtual network interface. This allows the device to appear as a network adapter to the host computer, enabling IP-based communication.

9.5.1 Setup

- Connect the device's USB gadget port to the PC using a USB cable capable of data transfer. Refer to TobuFi-DVK Hardware or Robonode Hardware for USB gadget port description and location on board.
- On your host computer, the device should appear as a new network interface:
- **Linux**: typically `usb0` or `enp...`

- **macOS**: typically `en<number>`
- **Windows**: appears as "Remote NDIS based Internet Sharing Device"
- The device's USB network interface typically uses a default IP address (e.g., 192.168.2.1). Check your device configuration for the actual address.

9.5.2 Usage

Once USB RNDIS connectivity is established, you can use it for:

- SSH connections
- GUI access

9.5.3 Troubleshooting

- **Windows**: May require RNDIS driver installation on first connection
- **Linux**: Interface may need to be manually brought up with `ifconfig usb0 up`
- **All platforms**: Ensure USB cable supports data transfer (not charge-only)

10. System Controls and Software Update

10.1 Overview

This section covers system control operations including software updates, device reboot, factory reset, and service management.

10.2 Software Version Information

10.2.1 Version Display

The `swinfo` utility displays comprehensive system information:

```
swinfo
```

Example output:

```
Board name: Robonode rev2.0
Radio ID: Premium
Radio features: 2-5GHz 2x4
Serial Number: 601821d6
SW partition: A
SW Version: 1.1.0
Image: 8dev-robo-image
Distro: 8dev-drone
Machine: robonode
Build hashes:
  meta-8dev: 48009c1
  meta-8dev-premium: 53040f4
```

Output fields:

- **Serial Number** - Device unique identifier
- **SW partition** - Currently active boot slot (A or B)
- **SW Version** - Installed software version
- **Image** - Image name and type
- **Distro** - Distribution name
- **Machine** - Target hardware platform
- **Build hashes** - Git commit hashes of build layers

10.2.2 Version Files

System version information is stored in multiple locations:

```
cat /etc/version                # System version
cat /lib/release/version        # Version string
cat /lib/release/build          # Build metadata
cat /lib/release/hashes         # Git commit hashes
```

10.3 Board and Radio Identification

10.3.1 boardinfo

The `boardinfo` command identifies the board hardware, radio type, and serial number by querying hardware identifiers.

Display board information:

```
boardinfo
```

Example output:

```
Board name: Robonode rev2.0
Radio ID: Premium
Radio features: 2-6GHz 2x4
Serial Number: 601821d6
```

Information provided:

- **Board name** — Identifies the board model and revision (TobuFi-DVK rev3/4/5, Robonode rev2)
- **Radio ID** — Identifies radio type: Standard or Premium
- **Radio features** — Frequency range and antenna configuration
- **Serial Number** — Device serial number from manufacturing

The `boardinfo` command reads board identity from the CDT partition and radio identity from PCI subsystem identifiers.

10.4 Software Update System

10.4.1 Update Framework

The system uses SWUpdate framework with the following characteristics:

- **A/B Partitioning** - Dual boot slots for redundant updates
- **Atomic Updates** - Complete installation or rollback, no partial states
- **Hardware Compatibility Validation** - Prevents installation of incompatible software
- **Automatic Rollback** - Reverts to previous version if new version fails to boot

10.4.2 Partition Scheme

The system maintains two complete boot environments:

Slot A:

- Boot partition (kernel image)
- System partition (root filesystem)

Slot B:

- Boot partition (kernel image)
- System partition (root filesystem)

Only one slot is active at boot time. The inactive slot is used for installing updates.

10.4.3 Update Process

Software update follows these steps:

- **Validation** - Hardware compatibility and version checks
- **Installation** - Update written to inactive partition (A→B or B→A)
- **Verification** - Integrity checks of installed components
- **Partition Switch** - System configured to boot from updated partition
- **Manual Reboot** - User reboots device to apply update
- **Boot Verification** - System boots from new partition
- **Automatic Rollback** - If boot fails 7 times, system reverts to previous partition

Important characteristics:

- Device remains operational during update (runs from current partition)
- Update does not interrupt running services
- Reboot is required to activate new software
- Power loss during installation may corrupt inactive partition but does not affect running system

10.4.4 Update Prerequisites

Before performing update:

- **Update Package** - Obtain `.swu` file for target hardware
- **Hardware Match** - Verify update package matches device machine type
- **Storage Space** - Verify sufficient space for installation
- **Power Supply** - Ensure stable power throughout update process

10.5 Software Update Methods

10.5.1 Local Update via uploaded file

Copy update file to device and apply locally:

```
# Copy update file to device
scp update-image-[variant].swu root@<device-ip>:/tmp/
# or via ADB
adb push update-image-[variant].swu /tmp/

# Apply update with progress display
update /tmp/update-image-[variant].swu

# Reboot to activate
reboot
```

Use case: Stable network connection, local development, manual control

10.5.2 update command

The `update` command provides a user-friendly software update with visual progress display. It uses `swupdate-client` and `swupdate-progress` under the hood and reboots the device to the new software after the operation completes.

Usage:

```
update <path-to-swu-file>
```

10.6 Verifying Software Update

10.6.1 Before Reboot

After update installation completes:

```
# Check installation logs
journalctl -u swupdate.service | tail -50

# Verify no errors reported
# Look for "Update successful" or similar messages
```

Note: Update logs are not preserved after reboot.

10.6.2 After Reboot

Verify successful update:

```
swinfo
```

Compare with pre-update information:

- **SW Version** - Should show new version
- **SW partition** - May have switched (A→B or B→A)
- **Build hashes** - Should match new software

10.7 Device Reboot

10.7.1 Reboot Command

Reboot the device:

```
reboot
```

Reboot process:

- Services stop gracefully
- Filesystems sync to storage
- Device restarts
- Services start according to boot sequence

Typical reboot time: 30-45 seconds

10.7.2 When to Reboot

Reboot is required or recommended for:

- **Activating software update** - Required after update installation
- **Applying configuration changes** - Some configuration changes require reboot
- **Recovering from service failures** - When multiple services fail
- **Clearing system state** - When system behaves unexpectedly

Note: Most configuration changes do not require reboot. Network and WiFi configuration typically applies through service restart.

10.8 Factory Reset

10.8.1 Factory Reset Command

Reset device to factory default configuration:

```
factory-reset
```

Without automatic reboot:

```
factory-reset -n
```

10.8.2 Factory Reset Actions

Factory reset performs the following:

- Removes all custom configuration changes
- Restores original factory configuration
- Clears configuration overlay on `/etc`
- Optionally reboots device

What is preserved:

- Installed software version (no downgrade)
- System partition contents

What is reset:

- System configuration in `/etc/roboconf/config.yaml`
- Other `/etc` contents modified after initial installation
- Service configurations derived from system configuration

10.8.3 After Factory Reset

After factory reset:

- Device boots with factory default configuration
- Default IP addresses are assigned
- Default WiFi settings are applied
- Web interface accessible at default IP
- Reconfiguration required for custom settings

10.8.4 Use Cases

Factory reset is useful for:

- **Troubleshooting configuration issues** - Remove problematic settings
- **Device repurposing** - Clear device-specific configuration
- **Recovery from misconfiguration** - Restore known-good configuration
- **Returning to known state** - Start from clean configuration

Note: Factory reset is not a security measure. It does not erase user data or prevent data recovery.

10.9 Service Management

Services are managed using systemd commands. Service restart is typically required after configuration changes.

10.9.1 Common Services

WiFi services:

```
systemctl restart hostapd@wlan0
systemctl restart wpa_supplicant@wlan1
```

Web server:

```
systemctl restart nginx
```

Spectral scan:

```
systemctl restart spectrald
```

WebSocket server:

```
systemctl restart websocket
```

WiFi Broadcast (per-stream instances):

```
systemctl restart wifibroadcast@tx
systemctl restart wifibroadcast@rx
```

Serial port forwarding (per-port instances):

```
systemctl restart ser2net@ttyMSM1
```

10.9.2 Service Status

Service status indicates:

- **active (running)** - Service is running normally
- **inactive (dead)** - Service is stopped
- **failed** - Service failed to start or crashed
- **activating** - Service is starting

Check service status:

```
systemctl status <service-name>
```

10.9.3 Applying Configuration Changes

After modifying configuration:

- Validate configuration: `config-validate`
- Restart affected services
- Verify services started successfully
- Check functionality

Example - apply WiFi configuration:

```
config-validate
systemctl restart hostapd@wlan0
systemctl status hostapd@wlan0
```

10.10 Update Troubleshooting

10.10.1 Hardware Incompatibility Error

Error message:

```
ERROR : HW compatibility failed
ERROR : Software image incompatible with hardware
```

Cause: Update package doesn't match target hardware

Resolution:

- Check device machine type: `swinfo | grep Machine`
- Verify update package is built for correct machine
- Obtain matching update package

10.10.2 Installation Timeout

Error message:

```
ERROR : Installation failed!
ERROR : Timeout waiting for device
```

Cause: Storage issues or corrupted package

Resolution:

- Check available storage: `df -h`
- Verify update package integrity
- Review installation logs: `journalctl -u swupdate.service | tail -50`
- Monitor logs in real-time: `journalctl -f -u swupdate`

10.10.3 Boot Failure After Update

Symptom: System fails to boot after reboot

Cause: Corrupted update or incompatible software

Resolution:

- Automatic rollback activates after 7 consecutive boot failures
- Wait 3-5 minutes for automatic rollback
- System reverts to previous working partition automatically
- No user intervention required

10.10.4 Update File Not Found

Error message:

```
ERROR : Cannot open /tmp/update-image-[variant].swu
ERROR : File not found
```

Cause: Incorrect file path or permissions

Resolution:

- Verify file exists: `ls -la /tmp/update-image-[variant].swu`
- Check file permissions (should be readable)
- Verify file transfer completed successfully

10.11 Next Steps

- Configuration - Configuration parameters description
- System Startup - System boot process and indicators
- Monitoring - System monitoring and diagnostics

11. Configuration Command-Line Tools

11.1 Overview

This section describes command-line utilities for configuration management. These tools provide configuration validation, application, and reset capabilities.

11.2 Configuration Validation

11.2.1 config-validate

Validates configuration file against schema.

Validate default configuration:

```
config-validate
```

Validates `/etc/roboconf/config.yaml` by default.

Validate specific file:

```
config-validate /path/to/config.yaml
```

Supported formats:

- YAML files (`.yaml`, `.yml`)
- JSON files (`.json`)

Exit codes:

- `0` - Configuration is valid
- `1` - Configuration has validation errors

Output: Validation errors are printed with specific error messages indicating the problem (invalid value, missing field, type mismatch, invalid reference, etc.).

11.3 Configuration Application

11.3.1 esconf reload

Applies configuration changes to system.

Command:

```
esconf reload
```

Process: 1. Reads `/etc/roboconf/config.yaml` 2. Generates service-specific configurations 3. Restarts affected services

Services reloaded: - Network manager (netman) - WiFi services (hostapd, wpa_supplicant) - Other services depending on configuration

Important: Configuration must be validated before applying.

11.4 Configuration Reset

11.4.1 esconf reset

Soft configuration reset to factory defaults. Restores configuration file to factory defaults and automatically reloads/applies it in system.

Command:

```
esconf reset
```

Process: 1. Resets `/etc/roboconf/config.yaml` to factory defaults 2. Automatically reloads and applies default configuration to system

Note: This performs soft configuration reset only. For complete factory reset including all system settings, see System Controls - Factory Reset.

11.5 Usage Workflows

11.5.1 Configuration Modification Workflow

Standard workflow for modifying configuration:

```
# 1. Edit configuration
vim /etc/roboconf/config.yaml

# 2. Validate changes
config-validate
```

```
# 3. If valid, apply configuration
esconf reload
```

11.6 Troubleshooting

11.6.1 Configuration Validation Fails

If `config-validate` reports errors:

- Review error messages carefully - they indicate specific problems
- Check field names for correct spelling and case (field names are case-sensitive)
- Verify all referenced zones exist in `network.zones`
- Ensure values are within allowed ranges (check parameter tables in Configuration)
- Verify YAML syntax:
- Use spaces for indentation (not tabs)
- Consistent indentation (2 spaces recommended)
- Quote strings containing special characters
- Boolean values: `true` or `false` (lowercase, unquoted)

11.6.2 Configuration Changes Not Applied

If configuration changes don't take effect:

- Validate configuration:

```
config-validate
```

- If validation passes, apply configuration:

```
esconf reload
```

- Check for service failures:

```
systemctl --failed
```

- Review service logs:

```
journalctl -u netman
```

- Check service status if needed:

```
systemctl status netman
systemctl status hostapd@wlan0
systemctl status wpa_supplicant@wlan0
```

11.7 Next Steps

- Configuration - Configuration parameters description
- Management REST API - Remote management over REST API
- System Controls - Device control operations

12. Configuration Parameters

12.1 Overview

This section describes the RoboConf configuration system and available configuration parameters. RoboConf uses YAML-based configuration with schema validation and automatic parameter backfilling.

12.2 Configuration System

12.2.1 Configuration Files

The system uses multiple configuration files:

File	Location	Purpose	Editable
Board Information	`/etc/board.conf`	Production data and device identity	No
Board File	`/etc/roboconf/board.yaml`	Hardware capabilities and constraints	No
Configuration File	`/etc/roboconf/config.yaml`	Active system configuration	Yes
Schema File	`/usr/share/roboconf/schema.yaml`	Validation rules	No
Default Configuration	`/usr/share/roboconf/default.yaml`	Factory defaults	No

12.2.2 Board Capabilities (`/etc/roboconf/board.yaml`)

Board file defines hardware capabilities and constraints. Configuration must comply with board file limits.

Board file contains:

- Radio frequency ranges and supported channels
- Maximum transmit power limits
- Supported WiFi protocols and bandwidths
- Available network interfaces
- Hardware-specific feature flags

Example board file content:

```
version: "1.1"
wifi:
  radio0:
    frequencies: [2300-3100, 4500-6100]
    bandwidth: [5, 10, 20, 40, 80, 160]
    txpower_max: 32
    modes: [BSS, NAW, WFB]
  radio1:
    frequencies: [2400-2500, 5150-5850]
    bandwidth: [20, 40, 80]
    txpower_max: 16
    modes: [BSS]
```

Board file is shipped with firmware and initialised for detected board.

12.2.3 Board Information (`/etc/board.conf`)

Board information file contains production data programmed during manufacturing.

File format: Shell-compatible variable definitions generated from EEPROM at boot.

Board information fields:

- `PRODUCT_ID` - Product identifier
- `PCB_REVISION` - PCB hardware revision
- `PCB_NAME` - PCB design name
- `PCB_SN` - PCB serial number
- `PCB_PROD_DATE` - Production date
- `PCB_PROD_LOCATION` - Production facility
- `SERIAL_NO` - Device serial number
- `MAC_ADDR_eth0` - Ethernet MAC address
- `MAC_ADDR_wlan0` - Pine WiFi MAC address
- `MAC_ADDR_wlan1` - Internal WiFi MAC address

Example board information (Robonode):

```
PRODUCT_ID=Robonode
PCB_REVISION=02
PCB_NAME=ROB_100
PCB_SN=1234567890
```

```
PCB_PROD_DATE=2025-10-13
PCB_PROD_LOCATION=Factory
SERIAL_NO=RN0987654321
MAC_ADDR_eth0=00:03:7F:12:90:8F
MAC_ADDR_wlan0=00:03:7F:12:90:90
MAC_ADDR_wlan1=00:03:7F:12:90:91
```

Example board information (TobuFi-DVK):

```
PRODUCT_ID=TobuFi-DVK
PCB_REVISION=01
PCB_NAME=TOB_200
PCB_SN=0987654321
PCB_PROD_DATE=2025-11-20
PCB_PROD_LOCATION=Factory
SERIAL_NO=DVK1234567890
MAC_ADDR_eth0=00:03:7F:12:A0:8F
MAC_ADDR_wlan0=00:03:7F:12:A0:90
MAC_ADDR_wlan1=00:03:7F:12:A0:91
```

Board information is read-only and generated automatically at boot from hardware EEPROM.

12.3 Configuration Structure

12.3.1 Version Field

Every configuration file must include version field:

```
version: "1.1"
```

The version identifies schema compatibility. Configuration with mismatched version may be rejected.

12.3.2 Network Zones

Network zones define logical network segments. Multiple interfaces can belong to the same zone.

Zone parameters:

Parameter	Type	Values	Description
`mode`	enum	`static`, `dhcp-client`	IP address assignment method
`address`	IPv4	Valid IPv4 address	Primary IP for this zone

Addressing modes:

- `static` - Fixed IP address assignment
- `dhcp-client` - Obtain IP from DHCP server. If DHCP fails, the configured address is used as fallback.

Example:

```
network:
  zones:
    data:
      mode: "static"
      address: 192.168.1.1
    mgmt:
      mode: "dhcp-client"
      address: 192.168.2.1
```

12.3.3 Ethernet Interfaces

Ethernet interface configuration assigns interfaces to network zones.

Interface parameters:

Parameter	Type	Description
`zone`	string	Network zone name (must exist in `network.zones`)

Example:

```
ethernet:
  eth0:
    zone: "data"
```

Zone inheritance: Interface inherits all settings from assigned zone (IP address, DHCP mode, etc.).

Automatic bridging: If multiple interfaces are assigned to same zone, bridge is created automatically (`br-<zone_name>`).

12.3.4 WiFi Configuration

WiFi configuration is organized by radio interface. The system supports two radios with different capabilities.

Radio Interfaces

Pine Radio (`radio0`) - External module with extended capabilities:

- Frequency range: 2.3-3.1 GHz, 4.5-6.1 GHz
- Bandwidth: 5, 10, 20, 40, 80, 160 MHz
- Modes: BSS, NAW, WFB
- Protocol: 802.11ax (WiFi 6)

Internal Radio (`radio1`) - Built-in standard WiFi:

- Frequency range: 2.4 GHz, 5 GHz standard bands
- Bandwidth: 20, 40, 80 MHz
- Modes: BSS only
- Protocol: 802.11ac (WiFi 5)

Radio Parameters

Pine Radio (radio0) parameters:

Parameter	Type	Range/Values	Description
`mode`	enum	`BSS` , `NAW` , `WFB`	Operation mode
`country`	string	2 characters	Country code for regulatory compliance
`channel`	int	0, 1-13, 36-165	WiFi channel (0 = auto)
`frequency`	int	0, 2360-3150, 4550-6630	Custom frequency in MHz (0 = use channel)
`width`	enum	5, 10, 20, 40, 80, 160	Channel bandwidth in MHz
`ack_timeout`	int	1-255	Acknowledgment timeout
`cca_threshold`	int	-100 to 0	CCA threshold in dBm
`tx_burst`	bool	true, false	WMM Tx burst for monitor interface
`txpower`	int	0-32	Transmission power in dBm
`proto`	enum	`ht` , `vht` , `he`	WiFi protocol (11n, 11ac, 11ax)
`gi`	enum	400, 800, 1600, 3200	Guard interval in nanoseconds
`mcs`	int/str	0-15, `auto`	Modulation and Coding Scheme
`nss`	int/str	1-2, `auto`	Number of Spatial Streams

Internal Radio (radio1) parameters:

Parameter	Type	Range/Values	Description
`mode`	enum	`BSS`	Operation mode (BSS only)
`country`	string	2 characters	Country code for regulatory compliance
`channel`	int	0, 1-13, 36-165	WiFi channel (0 = auto)
`width`	enum	20, 40, 80	Channel bandwidth in MHz
`txpower`	int	0-16	Transmission power in dBm
`proto`	enum	`ht` , `vht`	WiFi protocol (11n, 11ac)
`gi`	enum	400, 800	Guard interval in nanoseconds

WiFi Modes

BSS Mode (Basic Service Set): Standard WiFi mode supporting Access Point and Station operation.

When radio mode is `BSS`, configure `ap` (Access Point) and `sta` (Station) sub-sections.

NAW Mode (Non-Associated WiFi): Special operation mode available on Pine Radio only.

When radio mode is `NAW`, configure `naw` sub-section.

WFB Mode (WiFi Broadcast): WiFi Broadcast mode for unidirectional low-latency data streaming. Available on Pine Radio only.

When radio mode is `WFB`, the radio operates exclusively in monitor mode. Configure WFB streams in the `wfb` section. See WFB Streams Configuration for details.

Access Point Configuration

Access Point parameters (used when `mode: "BSS"`):

Parameter	Type	Range/Values	Description
`enabled`	boolean	true, false	Whether AP is active
`ssid`	string	1-32 characters	Network name
`passphrase`	string	empty or 8-63 chars	WiFi password (empty = open network)
`zone`	string	zone name	Network zone assignment

Example:

```
wifi:
  radio1:
    mode: "BSS"
    country: "LT"
    channel: 6
    width: 20
    txpower: 20
    proto: "vht"
    gi: 800
    ap:
      ssid: "MyNetwork"
      passphrase: "password123"
      enabled: true
      zone: "data"
    sta:
      ssid: ""
```

```
      passphrase: ""
      enabled: false
      zone: "data"
```

Station Configuration

Station parameters (used when `mode: "BSS"`):

Parameter	Type	Range/Values	Description
`enabled`	boolean	true, false	Whether station is active
`ssid`	string	1-32 characters	Network name to connect to
`passphrase`	string	empty or 8-63 chars	WiFi password
`zone`	string	zone name	Network zone assignment

NAW Configuration

NAW parameters (used when `mode: "NAW"`):

Parameter	Type	Range/Values	Description
`enabled`	boolean	true, false	Whether NAW is active
`link_id`	string	1-32 characters	NAW link identifier
`passphrase`	string	empty or 8-63 chars	Link password
`zone`	string	zone name	Network zone assignment

Example:

```
wifi:
  radio0:
    mode: "NAW"
    country: "LT"
    frequency: 5180
    width: 80
    txpower: 30
    proto: "he"
    gi: 800
    naw:
      link_id: "NAW-Link-1"
      passphrase: "nawpassword"
```

```
      enabled: true
      zone: "data"
```

Monitor Interface

The monitor interface is used for spectral scan and WFB mode operation.

Monitor parameters:

Parameter	Type	Range/Values	Description
`mon.enabled`	boolean	true, false	Enable monitor interface

Monitor interface is automatically enabled when radio mode is `WFB`. For spectral scan in `BSS` or `NAW` modes, enable the monitor interface explicitly.

Example:

```
wifi:
  radio0:
    mode: "BSS"
    mon:
      enabled: true
```

12.3.5 Spectral Service Configuration

The spectral service provides spectrum analysis capabilities for the Pine Radio.

Spectral parameters:

Parameter	Type	Range/ Values	Description
`spectral.enabled`	boolean	true, false	Enable spectral scan service
`spectral.bgscan`	boolean	true, false	Enable background spectral sweep (agile scan)

Background spectral sweep (agile scan) performs continuous frequency scanning in the background without interrupting normal radio operation. When disabled, spectral scan operates only on the current operating frequency.

Example:

```
spectral:
  enabled: true
  bgscan: false
```

12.3.6 WFB Streams Configuration

WiFi Broadcast (WFB) provides unidirectional low-latency data streaming using monitor mode. It maps UDP packets to raw WiFi frames with Forward Error Correction (FEC) for reliability over lossy wireless links.

Prerequisites:

- Pine Radio (`radio0`) mode must be set to `WFB`
- Monitor interface is automatically enabled in WFB mode

Stream parameters:

Parameter	Type	Range/Values	Description
`enabled`	bool	true, false	Enable this stream
`name`	string	0-16 characters	Stream display name
`mode`	enum	`tx`, `rx`	Stream direction
`stream_id`	int	0-255	Unique stream identifier
`listen_port`	int	1-65535	UDP port to listen for input data (TX mode)
`forward_port`	int	1-65535	UDP port to forward received data (RX mode)
`forward_addr`	IPv4	Valid IPv4 address	Destination address for received data (RX mode)
`fec_k`	int	0+	FEC data packets per block
`fec_n`	int	0+	FEC total packets per block (data + parity)
`proto`	enum	`ht`, `vht`, `he`	WiFi protocol override
`gi`	enum	400, 800, 1600, 3200	Guard interval override in nanoseconds
`mcs`	int/str	0-15, `auto`	MCS override
`nss`	int/str	1-2, `auto`	NSS override
`qdisc`	bool	true, false	Enable Linux traffic shaping

Stream modes:

- `tx` **(transmit):** Listens on `listen_port` for UDP input data and transmits it over air as raw WiFi frames.
- `rx` **(receive):** Receives data over air and forwards it to `forward_addr forward_port` as UDP packets.

FEC parameters:

- `fec_k` — number of data packets per FEC block
- `fec_n` — total packets per block (data + parity). The difference `fec_n - fec_k` is the number of parity packets.
- Default: k=8, n=12 — recovers up to 4 lost packets per block of 12.
- Set `fec_k: 0` and `fec_n: 0` to disable FEC.

Per-stream radio overrides:

Per-stream `proto`, `gi`, `mcs`, and `nss` parameters override the radio-level settings for that stream only.

Example:

```
wifi:
  radio0:
    mode: "WFB"
    country: "LT"
    frequency: 5180
    width: 20
    txpower: 20
    proto: "he"
    gi: 800
wfb:
  streams:
    tx:
      enabled: true
      mode: "tx"
      listen_port: 5600
      fec_k: 8
      fec_n: 12
      stbc: false
      ldpc: false
      qdisc: true
      stream_id: 1
    rx:
      enabled: false
      mode: "rx"
      forward_addr: 127.0.0.1
      forward_port: 5600
      stream_id: 1
```

12.3.7 UART Configuration

UART configuration provides serial port to network forwarding. Available serial ports and their names are defined in the board file.

Available ports:

Device	Board Name	Description
`ttyMSM1`	UART0	Serial port 0
`ttyMSM2`	UART1	Serial port 1
`ttyMSM3`	UART2	Serial port 2

UART parameters:

Parameter	Type	Range/Values	Description
`access_port`	int	1-65535	Network access port number
`serial_baud`	int	1+	Serial baud rate
`serial_frame`	string	regex `[neoms][5-8][12]`	Serial frame format (parity + data bits + stop bits)
`access_protocol`	enum	`tcp` , `udp`	Network access protocol
`udp_stream`	bool	true, false	Enable UDP streaming mode
`udp_stream_addr`	IPv4	Valid IPv4 address	UDP stream destination address
`udp_stream_port`	int	1-65535	UDP stream destination port

Serial frame format:

The `serial_frame` parameter is a 3-character string: `<parity><databits><stopbits>`

- Parity: `N` (none), `E` (even), `O` (odd), `M` (mark), `S` (space)
- Data bits: `5` , `6` , `7` , `8`
- Stop bits: `1` , `2`

Example: `N81` = No parity, 8 data bits, 1 stop bit

Access modes:

- **TCP server:** Accepts TCP connections on `access_port`. Connect with any TCP client to access serial port.
- **UDP server:** Accepts UDP packets on `access_port`. Send UDP data to interact with serial port.
- **UDP streaming:** When `udp_stream` is enabled, serial data is continuously forwarded to `udp_stream_addr udp_stream_port` as UDP packets.

Example:

```
uart:
  ttyMSM1:
    access_protocol: "tcp"
    access_port: 3001
    serial_baud: 115200
    serial_frame: "N81"
    udp_stream: false
    udp_stream_addr: 127.0.0.1
    udp_stream_port: 5001
  ttyMSM2:
    access_protocol: "tcp"
    access_port: 3002
    serial_baud: 115200
    serial_frame: "N81"
    udp_stream: false
    udp_stream_addr: 127.0.0.1
    udp_stream_port: 5002
  ttyMSM3:
    access_protocol: "tcp"
    access_port: 3003
    serial_baud: 115200
    serial_frame: "N81"
    udp_stream: false
    udp_stream_addr: 127.0.0.1
    udp_stream_port: 5003
```

12.4 Configuration Scenarios

12.4.1 Single Zone Network

All interfaces in one network segment.

Required configuration:

- Define single zone in `network.zones`
- Assign all interfaces (ethernet, WiFi) to that zone
- Configure WiFi radio in BSS mode with AP or station

12.4.2 Multi-Zone Network

Separate zones for different purposes (e.g., data and management).

Required configuration:

- Define multiple zones in `network.zones` with different addresses
- Assign ethernet to one zone
- Configure first radio for data network (assigned to data zone)
- Configure second radio for management network (assigned to mgmt zone)
- Use different SSIDs for each radio

12.4.3 NAW Operation

Non-Associated WiFi operation for specialized communication.

Required configuration:

- Configure Pine Radio (`radio0`) with `mode: "NAW"`
- Set custom frequency (not standard channel)
- Configure `naw` sub-section with link identifier and password
- Assign to appropriate network zone

12.4.4 WFB Operation

WiFi Broadcast operation for low-latency unidirectional streaming.

Required configuration:

- Configure Pine Radio (`radio0`) with `mode: "WFB"`
- Set frequency and channel width
- Configure WFB streams in `wfb.streams` section
- Each stream needs unique `stream_id` and appropriate `mode` (tx/rx)

12.5 Configuration Syntax

12.5.1 YAML Format

Configuration uses YAML syntax:

- Use spaces for indentation (not tabs)
- Consistent indentation (2 spaces recommended)
- Quote strings containing special characters
- Boolean values: `true` or `false` (lowercase, unquoted)
- Integers: unquoted numbers
- Strings: quoted or unquoted

12.5.2 Field Requirements

Case sensitivity: Field names are case-sensitive. Use exact names as documented.

Mode-specific sections:

- Configure `ap`/`sta` only when radio `mode: "BSS"`
- Configure `naw` only when radio `mode: "NAW"`
- Configure `wfb` streams when radio `mode: "WFB"`
- Internal radio (`radio1`) supports only `mode: "BSS"`

12.5.3 Configuration Validation

Configuration validation ensures system integrity:

- **Schema Validation** - Structure and type checking against validation rules
- **Constraint Validation** - Range limits and enum value verification
- **Cross-Reference Validation** - References to zones and other objects exist
- **Backfilling** - Missing parameters filled from device defaults

Invalid configuration is rejected and not applied to the system.

12.5.4 Configuration Hierarchy

Configuration merging follows this hierarchy:

- User configuration in `/etc/roboconf/config.yaml`
- Default values from `/usr/share/roboconf/default.yaml`
- Hardware constraints from `/etc/roboconf/board.yaml`

User values override defaults. Hardware constraints cannot be exceeded.

12.6 Next Steps

- Configuration Tools - Command-line configuration tools
- System Controls - System controls and software update
- Monitoring - System monitoring and diagnostics

13. Network Connectivity

13.1 Overview

This section describes how to configure WiFi connectivity modes and troubleshoot network connectivity issues. For detailed parameter descriptions, see Configuration.

13.2 WiFi Connectivity Modes

13.2.1 Access Point (AP) Mode

Access Point mode broadcasts a WiFi network for client connections.

How to configure:

- Set radio mode to `BSS`
- Configure `ap` sub-section:

```
wifi:
  radio0:
    mode: "BSS"
    ap:
      enabled: true
      ssid: "MyNetwork"
      passphrase: "password123"
      zone: "data"
    sta:
      enabled: false
```

- Apply configuration:

```
config-validate
esconf reload
```

- Verify AP is running:

```
systemctl status hostapd@wlan0
```

Security: - Empty passphrase creates open network - Passphrase 8-63 characters creates WPA2-PSK encrypted network

13.2.2 Station (STA) Mode

Station mode connects device to an existing WiFi network.

How to configure:

- Set radio mode to `BSS`
- Configure `sta` sub-section:

```
wifi:
  radio0:
    mode: "BSS"
    ap:
      enabled: false
    sta:
      enabled: true
      ssid: "ExistingNetwork"
      passphrase: "networkpassword"
      zone: "data"
```

- Apply configuration:

```
config-validate
esconf reload
```

- Verify station is connected:

```
systemctl status wpa_supplicant@wlan0
iw dev wlan0 link
```

13.2.3 NAW Mode (Non-Associated WiFi)

NAW mode provides specialized wireless communication. Available on Pine Radio (radio0) only.

How to configure:

- Set Pine Radio mode to `NAW`
- Configure `naw` sub-section:

```
wifi:
  radio0:
    mode: "NAW"
    frequency: 5180
    naw:
      enabled: true
      link_id: "NAW-Link-1"
      passphrase: "nawpassword"
      zone: "data"
```

- Apply configuration:

```
config-validate
esconf reload
```

Note: NAW mode requires fixed channel / frequency to be set up.

13.2.4 WFB Mode (WiFi Broadcast)

WFB mode provides unidirectional low-latency data streaming over WiFi. Available on Pine Radio (radio0) only. The radio operates exclusively in WFB mode — AP and STA modes are not available simultaneously.

How to configure:

- Set Pine Radio mode to `WFB`
- Configure WFB streams:

```
wifi:
  radio0:
    mode: "WFB"
    country: "LT"
    frequency: 5180
    width: 20
    txpower: 20
    proto: "he"
    gi: 800
wfb:
  streams:
    tx:
      enabled: true
      mode: "tx"
      listen_port: 5600
      fec_k: 8
      fec_n: 12
      stream_id: 1
```

- Apply configuration:

```
config-validate
esconf reload
```

Note: WFB mode requires fixed frequency to be set. The radio does not provide standard WiFi service (AP/STA) while in WFB mode. See Configuration for detailed stream parameters.

13.2.5 Simultaneous AP and Station

Device can operate AP and Station modes simultaneously on different radios.

How to configure:

Configure one radio in AP mode and another in Station mode:

```
wifi:
  radio0:
    mode: "BSS"
    ap:
      enabled: true
      ssid: "MyAccessPoint"
      zone: "data"
```

```
    sta:
      enabled: false
  radio1:
    mode: "BSS"
    ap:
      enabled: false
    sta:
      enabled: true
      ssid: "UpstreamNetwork"
      zone: "data"
```

Use case: Device acts as WiFi repeater or provides local network while connected to upstream network.

13.2.6 DHCP Client Fallback

When a network zone is configured with `mode: "dhcp-client"`, the system first attempts to obtain an IP address via DHCP. If DHCP fails (no DHCP server available), the configured static `address` is used as a fallback. When DHCP later succeeds, the fallback address is removed and the DHCP-assigned address is used.

Example:

```
network:
  zones:
    data:
      mode: "dhcp-client"
      address: 192.168.1.1
```

In this example, the system attempts DHCP on the `data` zone. If no DHCP server responds, `192.168.1.1` is assigned as the fallback address. Once a DHCP server is found, the fallback address is replaced with the DHCP lease.

Use case: Ensures device is always reachable at a known IP address even when no DHCP server is available on the network.

13.3 Troubleshooting WiFi Connectivity

13.3.1 Access Point Not Visible

If WiFi Access Point network is not broadcasting:

- Verify AP is enabled:
- Check `wifi.radioX.ap.enabled: true`
- Verify SSID is configured
- Check radio configuration:
- Verify country code is set correctly
- Check channel is valid for country
- Ensure txpower is not set to 0
- Check hostapd service:

```
systemctl status hostapd@wlan0
journalctl -u hostapd@wlan0
```

Look for regulatory domain errors or channel conflicts.

- Verify radio is operational:

```
wifistats
iw dev
```

13.3.2 Station Cannot Connect to Network

If device cannot connect to WiFi network in Station mode:

- Verify station configuration:
- Check `wifi.radioX.sta.enabled: true`
- Verify SSID matches target network exactly (case-sensitive)
- Check passphrase is correct
- Check wpa_supplicant service:

```
systemctl status wpa_supplicant@wlan0
journalctl -u wpa_supplicant@wlan0
```

Look for authentication failures or connection errors.

- Verify network is in range:

```
iw dev wlan0 scan | grep -A 5 "SSID: YourNetworkName"
```

- Check connection status:

```
iw dev wlan0 link
```

13.3.3 WiFi Not Available After Boot

If WiFi interfaces are not working after boot:

- Review configuration in `/etc/roboconf/config.yaml`:
- Verify radio mode is set (BSS, NAW, or WFB)
- Check AP/STA/NAW sub-sections are configured
- Ensure zone assignment is correct
- Check WiFi interface status:

```
iw dev
```

- Check WiFi services:

```
systemctl status hostapd@wlan0
systemctl status wpa_supplicant@wlan0
```

- Review service logs:

```
journalctl -u hostapd@wlan0
journalctl -u wpa_supplicant@wlan0
```

- Check WiFi statistics:

```
wifistats
```

13.3.4 Network Not Available After Boot

If network interfaces are not working after boot:

- Review configuration in `/etc/roboconf/config.yaml`:
- Verify network zones are defined
- Check interfaces are assigned to zones
- Ensure zone addressing mode is correct
- Verify interface status:

```
ip addr show
```

- Check physical connections (ethernet):
- Verify cable is connected
- Check link status LEDs
- Check network service:

```
systemctl status netman
journalctl -u netman
```

13.4 Next Steps

- Configuration - Configuration parameters description
- Monitoring - System monitoring and diagnostics
- System Startup - System boot process and indicators

14. System Monitoring

14.1 Overview

This section describes system monitoring capabilities and diagnostic tools for the Robosoft platform. The system provides comprehensive monitoring tools for WiFi status, system resources, service health, and diagnostic data collection.

14.2 WiFi Monitoring

14.2.1 WiFi Statistics Tool

The `wifistats` command provides comprehensive WiFi radio and network monitoring.

Display WiFi status:

```
wifistats
```

Save WiFi status to JSON file:

```
wifistats --json /tmp/wifi_status.json
```

14.2.2 WiFi Statistics Information

Radio statistics include:

- Physical radio capabilities and configuration
- Operating frequency and channel width
- Transmit power and antenna configuration
- Protocol support (802.11a/b/g/n/ac/ax)

Network statistics include:

- Interface configuration and operational mode
- Security settings and authentication methods
- SSID configuration and network parameters

Example output:

```
Radio: wlan0
  PHY: phy0 (wiphy0)
  MAC: 00:11:22:33:44:55
  Country: US
  Frequency: 2437 MHz (Channel 6)
  Channel Width: 20 MHz
  TX Power: 20 dBm
  TX Chains: 2, RX Chains: 2
  Protocols: 802.11bgn

VAP: wlan0
  Interface: wlan0
  MAC: 00:11:22:33:44:56
  Mode: AP
  SSID: MyNetwork
  Security: WPA2-PSK/CCMP
```

14.2.3 Understanding WiFi Security Information

WPA Versions:

- `WPA` - WPA (802.11i draft)
- `WPA2` - WPA2 (802.11i-2004)
- `WPA3` - WPA3 (802.11i-2016)

Authentication Methods:

- `PSK` - Pre-shared key
- `8021X` - 802.1X enterprise
- `SAE` - Simultaneous Authentication of Equals (WPA3)

Cipher Suites:

- `CCMP` - AES-CCMP (preferred)
- `TKIP` - TKIP (legacy)
- `GCMP` - Galois/Counter Mode Protocol

14.3 System Diagnostics

14.3.1 Quick System Status

The `sysinfo` utility provides quick system status overview.

Display system information:

```
sysinfo
```

Example output:

```
MACHINE=Robonode
SW_VERSION=1.2.3
MEMORY=524288/1048576
PROCESSES=127
UNAME=Linux robonode 5.4.0 #1 SMP armv7l GNU/Linux
UPTIME=3600.45
CPU_LOAD=0.25
```

Note: On TobuFi-DVK boards, the output shows `MACHINE=TobuFi-DVK` instead of `MACHINE=Robonode`.

Information provided:

- Hardware platform and machine type
- Installed software version
- Memory usage (free/total in KB)
- Number of running processes
- System kernel information
- System uptime in seconds
- CPU load average

14.3.2 Comprehensive Diagnostic Collection

The `trouble` utility collects comprehensive diagnostic data for troubleshooting.

Create diagnostic archive:

```
trouble
```

Creates timestamped archive: `/tmp/device-MAC-YYYYMMDD_HHMMSS.zip`

Create diagnostic archive at specific location:

```
trouble /path/to/diagnostics.zip
```

Create uncompressed diagnostic directory:

```
trouble -d /path/to/diagnostics-directory
```

14.3.3 Diagnostic Archive Contents

The diagnostic archive includes:

System information:

- System date and time
- Kernel messages (dmesg)
- System journal logs
- Process list
- System kernel parameters

Network information:

- Network interface configuration
- IP address information
- Routing table
- Socket statistics

Resource information:

- Disk usage
- CPU information
- Memory information

Configuration files:

- User accounts
- System version
- Configuration files from `/etc`

14.3.4 Using Diagnostic Data

Collect diagnostic data when:

- Experiencing persistent issues
- Before contacting support
- After configuration changes that cause problems
- When services fail unexpectedly
- For documentation and analysis

Transfer diagnostic archive:

```
# Via SCP
scp root@<device-ip>:/tmp/diagnostics.zip /local/path/

# Via ADB
adb pull /tmp/diagnostics.zip /local/path/
```

14.4 Statistics Files

The system continuously records monitoring data to `/var/stats/`. Statistics are available in JSON format (latest snapshot) and text/CSV format (continuous log).

14.4.1 Spectral Scan

The spectral scan service provides real-time spectrum analysis for the Pine Radio. Enable it with `spectral.enabled: true` in configuration. When background scan is enabled (`spectral.bgscan: true`), the service performs continuous agile sweep across frequencies without interrupting normal radio operation.

Spectral data is also available in real-time through the web interface spectrum analyzer view with fullscreen mode and frequency markers.

CSV format (`/var/stats/spectrald-<timestamp>.csv`) — Continuous spectral scan log with one sample per row. Files are named with a timestamp (e.g., `spectrald-20260211-150135.csv`) and rotated periodically with older files compressed to `.csv.gz`:

```
timestamp,detector,freq_mhz,bandwidth_mhz,noise_dbm,rssi,bin_count,<spectral bins...>
1378763680,1,2412,20,-109,22,16,-102.64,-96.62,-96.62,-96.62,-106.5,-96.62,-82.49,-80.12,...
1378364284,1,2432,20,-109,18,16,-100.48,-109.0,-109.0,-99.02,-105.6,-103.48,-100.48,-94.19,..
1378416350,1,2472,20,-109,20,16,-99.83,-89.32,-95.34,-99.65,-102.55,-107.76,-108.55,-108.57,.
```

CSV columns include metadata fields (timestamp, detector, center frequency, bandwidth, noise floor, RSSI, bin count) followed by per-bin power values in dBm. The number of trailing columns matches `bin_count`.

14.4.2 WFB Stream Statistics

WFB stream statistics are recorded separately for TX and RX streams. Files are named `wfb-<profile>-<mode>.{json,txt}` based on the stream profile name and direction.

TX statistics (`/var/stats/wfb-tx-tx.json`) — Latest TX stream snapshot:


```
{
    "total": {
        "in_timeout": 0,
        "in_packets": 3750,
        "in_bytes": 394917,
        "tx_packets": 9197,
        "tx_bytes": 1149170,
        "tx_dropped": 0,
        "tx_truncated": 0
    },
    "timestamp": 1770817933.18,
    "in_timeout": 0,
    "in_packets": 2,
    "in_bytes": 117,
    "tx_packets": 5,
    "tx_bytes": 434,
    "tx_dropped": 0,
    "tx_truncated": 0
}
```


The `total` object contains cumulative counters since service start. The top-level fields contain counters for the latest reporting interval.

TX log (`/var/stats/wfb-tx-tx.txt`) — Continuous TX statistics:

```
# TS         Packets  Bytes      Drop     Truncate Timeout  InP      InB
33.4         7        1184       0        0        0        3        464
34.4         4        472        0        0        0        2        180
35.4         9        1058       0        0        0        4        373
```

RX statistics (`/var/stats/wfb-rx-rx.json`) — Latest RX stream snapshot:


```
{
    "total": {
        "rx_packets_all": 0,
        "rx_bytes_all": 0,
        "rx_corrupt": 0,
```

```
        "rx_error": 0,
        "rx_packets": 0,
        "rx_recovered": 0,
        "rx_lost": 0,
        "rx_invalid": 0,
        "out_packets": 0,
        "out_bytes": 0
    },
    "timestamp": 1770817920.31,
    "rx_packets_all": 0,
    "rx_bytes_all": 0,
    "rx_corrupt": 0,
    "rx_error": 0,
    "rx_packets": 0,
    "rx_recovered": 0,
    "rx_lost": 0,
    "rx_invalid": 0,
    "out_packets": 0,
    "out_bytes": 0
}
```


RX log (`/var/stats/wfb-rx-rx.txt`) — Continuous RX statistics:

```
# TS         Packets  Bytes      Good     Recover  Lost     Invalid  Errors   Corrupt
OutP     OutB       RSSI        SNR
33.2         0        0          0        0        0        0        0        0
0        0
```

14.4.3 WiFi Statistics

The `wifistats` utility collects WiFi radio and VAP status information via the kernel nl80211 interface.

JSON format (`/var/stats/wifistats.json`) — Current WiFi radio and VAP state:


```
{
  "radios": {
    "wlan0": {
      "wiphy_name": "phy0",
      "mac": "00:11:22:33:44:55",
      "country_code": "US",
      "frequency": 2437,
      "channel_width": 20,
      "tx_power": 20,
      "tx_chains": 2,
      "rx_chains": 2,
      "protocols": ["802.11b", "802.11g", "802.11n"]
    }
  },
  "vaps": {
```

```
    "wlan0": {
      "mac": "00:11:22:33:44:56",
      "mode": "AP",
      "ssid": "MyNetwork",
      "wpa_version": "WPA2",
      "akm_suite": "PSK",
      "pairwise_cipher": "CCMP",
      "group_cipher": "CCMP"
    }
  }
}
```

The `radios` section contains physical radio information (frequency, power, antenna chains, protocols). The `vaps` section contains virtual access point details (mode, SSID, security settings).

14.5 Service Monitoring

14.5.1 Service Status

Service status indicates operational state.

Check service status:

```
systemctl status <service-name>
```

Service states:

- `active (running)` - Service is running normally
- `inactive (dead)` - Service is stopped
- `failed` - Service failed to start or crashed
- `activating` - Service is starting

Check all failed services:

```
systemctl --failed
```

14.5.2 Key Services

Network manager:

```
systemctl status netman
```

WiFi services:

```
systemctl status hostapd@wlan0
systemctl status wpa_supplicant@wlan1
```

Web server:

```
systemctl status nginx
```

Spectral scan:

```
systemctl status spectrald
```

WebSocket server:

```
systemctl status websocket
```

WiFi Broadcast (per-stream instances):

```
systemctl status wifibroadcast@tx
systemctl status wifibroadcast@rx
```

Serial port forwarding (per-port instances):

```
systemctl status ser2net@ttyMSM1
systemctl status ser2net@ttyMSM2
systemctl status ser2net@ttyMSM3
```

14.6 Log Analysis

14.6.1 System Logs

System logs are managed by journald and accessible through `journalctl` command.

View current boot logs:

```
journalctl -b
```

View previous boot logs:

```
journalctl -b -1
```

View kernel messages:

```
dmesg
```

Search for errors in logs:

```
journalctl -b | grep -i error
```

14.6.2 Service Logs

View service logs:

```
journalctl -u <service-name>
```

View service logs in real-time:

```
journalctl -f -u <service-name>
```

View logs with line limit:

```
journalctl -u <service-name> | tail -50
```

Common services to monitor:

```
journalctl -u netman
journalctl -u hostapd@wlan0
journalctl -u wpa_supplicant@wlan1
```

14.6.3 Log Analysis

Logs help identify:

- Hardware detection issues
- Service startup failures
- Configuration errors
- Network initialization problems
- Software update status

Look for `[FAILED]` or `ERROR` messages in logs to identify issues.

14.7 Network Monitoring

14.7.1 Network Interface Status

List all network interfaces:

```
ip addr show
```

Verify expected interfaces are present and have IP addresses assigned.

Check WiFi interfaces:

```
iw dev
```

Verify WiFi interfaces are present and in correct mode (AP, station, etc.).

Check network connectivity:

```
ping <ip-address>
```

14.7.2 Bridge Status

List all bridges:

```
ip link show type bridge
```

Bridges are automatically created when multiple interfaces belong to the same network zone (format: `br-<zone_name>`).

Show bridge members:

```
bridge link show
```

14.8 Performance Monitoring

14.8.1 CPU and Memory

Check CPU and memory usage:

```
top
```

Check memory details:

```
free -h
```

List processes by memory usage:

```
ps aux --sort=-rss
```

14.8.2 Storage

Check disk usage:

```
df -h
```

Check directory sizes:

```
du -sh /path/to/directory
```

14.9 Troubleshooting WiFi Monitoring

14.9.1 Missing WiFi Information in Monitoring

If `wifistats` shows incomplete information:

Missing radio information:

- Driver not loaded - Check `dmesg | grep -i wifi` for driver errors
- Hardware not detected - Verify radio is properly connected
- Regulatory restrictions - Set correct country code

Missing network information:

- Interface not configured - Check configuration and apply with `esconf reload`
- hostapd/wpa_supplicant not running - Check service status
- Permission issues - Run as root or check service permissions

Incomplete security information:

- Interface not fully initialized - Wait for service startup to complete
- Unsupported security configuration - Check WPA version compatibility
- Driver limitations - Verify driver supports configured security mode

14.10 Next Steps

- System Startup - System boot process and indicators
- System Controls - System controls and software update
- Configuration - Configuration parameters description
- Configuration Tools - Command-line configuration tools
- Network Connectivity - Network and Wi-Fi configuration scenarios

15. Management REST API

15.1 Overview

This section describes REST API for remote system management and configuration control. The REST API is used by the web management interface and can be accessed by external applications.

15.2 API Base URL

```
http://<device-ip>/apiv1/
```

15.3 Authentication

Authentication requirements depend on web server configuration. Typically accessed within trusted network without authentication.

15.4 Response Format

All API responses use JSON format.

Success response:

```
{
  "error": false,
  "message": "Operation description",
  "data": { ... }
}
```

Error response:

```
{
  "error": true,
  "message": "Error description"
}
```

15.5 Configuration Endpoints

15.5.1 Get Configuration

Retrieve current system configuration.

Endpoint: `GET /apiv1/config`

Response: Configuration object in JSON format

Status codes: - `200` - Success - `500` - Configuration read error

Example response:

```
{
  "version": "1.1",
  "network": {
    "zones": {
      "data": {
        "mode": "static",
        "address": "192.168.1.1"
      }
    }
  },
  "ethernet": {
    "eth0": {
      "zone": "data"
    }
  },
  "wifi": {
    "radio0": {
      "mode": "BSS",
      "country": "LT",
      ...
    }
  },
  "wfb": {
    "streams": {
      "tx": { ... },
      "rx": { ... }
    }
  },
  "spectral": {
    "enabled": true,
    "bgscan": false
  },
  "uart": {
    "ttyMSM1": { ... },
    "ttyMSM2": { ... },
    "ttyMSM3": { ... }
  }
}
```

15.5.2 Update Configuration

Write new configuration to system.

Endpoint: `POST /apiv1/config`

Request body:

```
{
  "data": {
    "version": "1.1",
    "network": { ... },
    "ethernet": { ... },
    "wifi": { ... }
  }
}
```

Process:

- Validates configuration against schema
- Writes configuration to `/etc/roboconf/config.yaml`
- Executes `esconf reload` to apply changes
- Returns updated configuration

Response: Updated configuration object

Status codes:

- `200` - Success
- `400` - Invalid request payload or validation failed
- `500` - Configuration write failed or apply timeout

15.5.3 Reset Configuration

Reset configuration to factory defaults.

Endpoint: `POST /apiv1/reset`

Request body: Empty or `{}`

Process:

- Executes `esconf reset`
- Resets `/etc/roboconf/config.yaml` to defaults
- Automatically applies default configuration

Response:

```
{
  "error": false,
  "message": "Resetting device configuration file..."
}
```

Status codes:

- `200` - Success
- `500` - Configuration reset failed

15.6 Board Information Endpoints

15.6.1 Get Board Information

Retrieve hardware capabilities and specifications.

Endpoint: `GET /apiv1/board`

Response: Board information object

Status codes:

- `200` - Success
- `500` - Board information read error

Example response:

```
{
  "version": "1.1",
  "board": "robonode",
  "ethernet": {
    "eth0": {
      "speed": [100, 1000],
      "connector": "RJ45"
    }
  },
  "wifi": {
    "radio0": {
      "name": "Radio #0",
      "slug": "pine",
      "protocol": "11ax",
      "antennas": 4,
      "bands": [...]
    }
```

```
    }
}
```

15.7 System Control Endpoints

15.7.1 Reboot Device

Initiate device reboot.

Endpoint: `POST /apiv1/reboot`

Request body: Empty or `{}`

Process:

- Responds with success message
- Device reboots after 1 second delay

Response:

```
{
  "error": false,
  "message": "Device is rebooting..."
}
```

Status codes:

- `200` - Success (reboot initiated)

15.8 Software Update Endpoints

15.8.1 Upload Update File

Upload software update package to device.

Endpoint: `PUT /apiv1/update`

Request: Multipart form data with file field

Process:

- Receives `.swu` file
- Saves to `/var/volatile/tmp/firmware.swu`

Response:

```
{
  "error": false,
  "message": "Firmware image successfully transferred"
}
```

Status codes:

- `200` - Success
- `400` - Missing file, empty filename, or save failed

15.8.2 Apply Update

Apply uploaded software update.

Endpoint: `POST /apiv1/update`

Request body: Empty or `{}`

Process:

- Executes `swupdate-client` with uploaded file
- Reboots device if update succeeds

Response:

```
{
  "error": false,
  "message": "Device rebooting after upgrade..."
}
```

Status codes:

- `200` - Success (update applied, rebooting)
- `400` - Update installation failed

15.8.3 Delete Update File

Delete uploaded update file without applying.

Endpoint: `DELETE /apiv1/update`

Request body: Empty or `{}`

Process: Removes `/var/volatile/tmp/firmware.swu`

Response:

```
{
  "error": false,
  "message": "Firmware file deleted successfully"
}
```

Status codes:

- `200` - Success
- `400` - File deletion failed

15.9 Status Endpoints

15.9.1 Get System Status

Retrieve system status information.

Endpoint: `GET /apiv1/status`

Response:

```
{
  "host": "192.168.1.1"
}
```

Returns IP address of data bridge interface.

Status codes:

- `200` - Success

15.10 Usage Examples

15.10.1 Example 1: Retrieve and Modify Configuration

Using curl to get configuration, modify, and update:

```
# Get current configuration
curl http://192.168.1.1/apiv1/config > config.json
```

```
# Edit config.json with desired changes
vim config.json

# Update configuration
curl -X POST http://192.168.1.1/apiv1/config \
  -H "Content-Type: application/json" \
  -d @config.json
```

15.10.2 Example 2: Change WiFi SSID

Modify WiFi Access Point SSID programmatically:

```
# Get current config
CONFIG=$(curl -s http://192.168.1.1/apiv1/config)

# Modify SSID (using jq)
MODIFIED=$(echo "$CONFIG" | jq '.wifi.radio0.ap.ssid = "NewSSID"')

# Update configuration
curl -X POST http://192.168.1.1/apiv1/config \
  -H "Content-Type: application/json" \
  -d "{\"data\": $MODIFIED}"
```

15.10.3 Example 3: Software Update via API

Upload and apply software update:

```
# Upload update file
curl -X PUT http://192.168.1.1/apiv1/update \
  -F "file=@update-image-robox.swu"

# Apply update (device will reboot)
curl -X POST http://192.168.1.1/apiv1/update
```

15.10.4 Example 4: Reset Configuration

Reset to factory defaults via API:

```
# Reset configuration (automatically applied)
curl -X POST http://192.168.1.1/apiv1/reset

# Optionally reboot device
curl -X POST http://192.168.1.1/apiv1/reboot
```

15.11 Troubleshooting

15.11.1 API Configuration Update Fails

If REST API POST `/apiv1/config` returns errors:

Status 400 - Invalid request or validation failed:

- Check request body format (must be `{"data": {...}}`)
- Verify configuration passes schema validation
- Review error message in response

Status 500 - Configuration write or apply failed:

- Check available storage: `df -h`
- Review system logs: `journalctl -u netman`
- Verify services can restart successfully
- Check for permission issues

15.12 Next Steps

- Configuration - Configuration parameters description
- Configuration Tools - Command-line configuration tools
- System Controls - System controls and software update

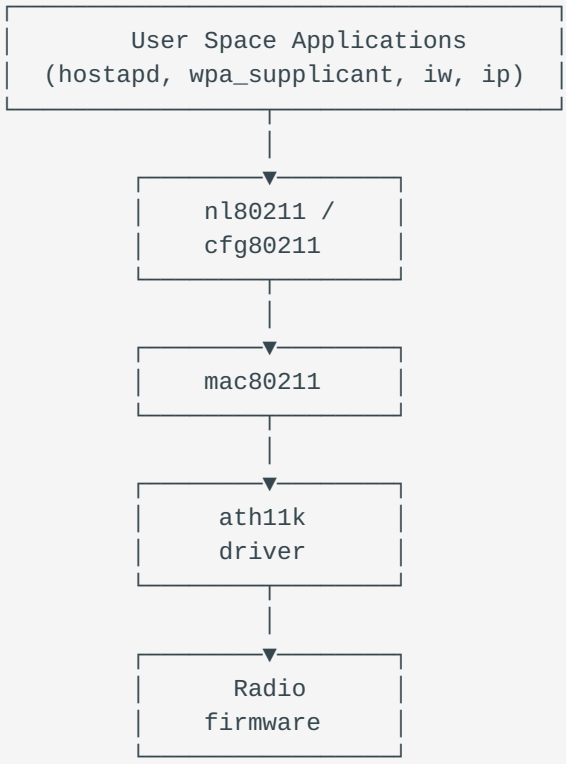
16. WiFi Controls

16.1 Overview

This guide describes generic WiFi radio control commands using standard Linux wireless tools. These commands work across different WiFi drivers and are essential for interface management, configuration, and monitoring.

For ath11k-specific advanced controls, see ATH11K Controls.

16.2 WiFi Stack Architecture



Component Responsibilities:

- **nl80211/cfg80211**: Kernel-to-userspace communication via netlink sockets, provides generic WiFi configuration API
- **mac80211**: Software MAC layer handling common 802.11 operations (frame handling, rate control, power save)
- **ath11k**: Hardware-specific driver implementation for Qualcomm WiFi chipsets
- **firmware**: Physical radio performing RF operations

16.3 iw Tool Commands

The `iw` utility is the standard Linux tool for WiFi configuration and management.

16.3.1 Interface Management

Create and manage virtual WiFi interfaces on physical radios:

```
# Create virtual interface on PHY
iw phy phy0 interface add wlan0 type __ap      # Access Point
iw phy phy0 interface add wlan1 type managed   # Station (client)
iw phy phy0 interface add wlan2 type mp        # Mesh Point

# Remove interface
iw dev wlan0 del

# Change interface type
iw dev wlan0 set type __ap
iw dev wlan0 set type managed
iw dev wlan0 set type mp
```

Interface Types:

- `__ap` - Access Point mode
- `managed` - Station/client mode
- `mp` - Mesh Point (802.11s)
- `monitor` - Monitor mode (packet capture)
- `adhoc` - Ad-hoc (IBSS) mode

16.3.2 Power Control

Set transmit power limits:

```
# Set TX power (in 0.01 dBm units)
iw dev wlan0 set txpower limit 2000    # 20 dBm
iw dev wlan0 set txpower limit 3000    # 30 dBm

# Set TX power in mBm (alternative)
iw dev wlan0 set txpower fixed 2000    # 20 dBm (fixed)
iw dev wlan0 set txpower auto          # Automatic power control
```

Power Level Conversion:

- 1 dBm = 100 units
- 20 dBm = 2000 units
- 30 dBm = 3000 units

16.3.3 WDS Configuration

Enable 4-address mode for wireless bridging (WDS - Wireless Distribution System):

```
# Enable 4-address mode for wireless bridging
iw dev wlan0 set 4addr on
iw dev wlan0 set 4addr off

# Bridge the interface after enabling WDS
ip link set wlan0 master br0
```

Note: Interface must be added to a bridge for WDS to function properly.

16.3.4 Information Queries

Get interface and radio capabilities:

```
# Get interface details
iw dev wlan0 info

# Get PHY capabilities
iw phy phy0 info

# List all WiFi interfaces
iw dev

# List all PHYs
iw list

# Get current channel and frequency
iw dev wlan0 info | grep channel

# Scan for available networks (station mode)
iw dev wlan0 scan
```

16.3.5 Channel and Frequency

Set operating channel and frequency:

```
# Set channel (when not managed by hostapd/wpa_supplicant)
iw dev wlan0 set channel 36

# Set frequency in MHz
iw dev wlan0 set freq 5180

# Set channel width
iw dev wlan0 set freq 5180 HT20        # 20 MHz
iw dev wlan0 set freq 5180 HT40+       # 40 MHz (upper)
iw dev wlan0 set freq 5180 HT40-       # 40 MHz (lower)
iw dev wlan0 set freq 5180 80MHz       # 80 MHz
```

Channel Planning Reference:

Band	Channel	Frequency (MHz)	DFS Required	Notes
2.4 GHz	1	2412	No	Non-overlapping
2.4 GHz	6	2437	No	Non-overlapping
2.4 GHz	11	2462	No	Non-overlapping
5 GHz	36	5180	No	UNII-1
5 GHz	40	5200	No	UNII-1
5 GHz	44	5220	No	UNII-1
5 GHz	48	5240	No	UNII-1
5 GHz	52	5260	Yes	UNII-2A
5 GHz	56	5280	Yes	UNII-2A
5 GHz	60	5300	Yes	UNII-2A
5 GHz	64	5320	Yes	UNII-2A
5 GHz	100	5500	Yes	UNII-2C
5 GHz	104	5520	Yes	UNII-2C
5 GHz	108	5540	Yes	UNII-2C
5 GHz	112	5560	Yes	UNII-2C
5 GHz	116	5580	Yes	UNII-2C
5 GHz	120	5600	Yes	UNII-2C
5 GHz	124	5620	Yes	UNII-2C
5 GHz	128	5640	Yes	UNII-2C
5 GHz	132	5660	Yes	UNII-2C
5 GHz	136	5680	Yes	UNII-2C
5 GHz	140	5700	Yes	UNII-2C
5 GHz	144	5720	Yes	UNII-2C
5 GHz	149	5745	No	UNII-3
5 GHz	153	5765	No	UNII-3

Band	Channel	Frequency (MHz)	DFS Required	Notes
5 GHz	157	5785	No	UNII-3
5 GHz	161	5805	No	UNII-3
5 GHz	165	5825	No	UNII-3

DFS (Dynamic Frequency Selection): Channels 52-144 require radar detection. The radio will perform Channel Availability Check (CAC) for 60 seconds before allowing transmissions. If radar is detected, the radio must vacate the channel within 10 seconds.

Channel Width Selection:

- **20 MHz**: Maximum compatibility, lowest throughput
- **40 MHz**: Good balance, suitable for most deployments
- **80 MHz**: High throughput, requires clear spectrum
- **160 MHz**: Maximum throughput, limited channel availability

16.3.6 Station information

```
# List all connected stations with statistics
iw dev wlan0 station dump

# Get specific station info
iw dev wlan0 station get <client_mac>

# Parse station signal strength
iw dev wlan0 station dump | grep -E "Station|signal:" | paste - -

# Count connected clients
iw dev wlan0 station dump | grep "^Station" | wc -l
```

16.4 Linux Network Interface Commands

Standard Linux networking commands for WiFi interface control.

16.4.1 Interface Control

Bring interfaces up and down:

```
# Bring interface up
ip link set wlan0 up
```

```
# Bring interface down
ip link set wlan0 down

# Check interface status
ip link show wlan0
```

16.4.2 Bridge Management

Add/remove interfaces from bridges:

```
# Add interface to bridge
ip link set wlan0 master br0

# Remove interface from bridge
ip link set wlan0 nomaster

# Show bridge members
ip link show master br0

# Show which bridge an interface belongs to
ip link show wlan0 | grep master
```

16.4.3 Hardware Information

Query interface hardware details:

```
# Get MAC address
cat /sys/class/net/wlan0/address
ip link show wlan0 | grep link/ether

# Check if interface exists
ls /sys/class/net/wlan0/ 2>/dev/null && echo "Interface exists"

# Get interface index
cat /sys/class/net/wlan0/ifindex

# Get interface carrier status
cat /sys/class/net/wlan0/carrier

# Get PHY index
cat /sys/class/net/wlan0/phy80211/index
```

16.4.4 MAC Address Control

Set or change MAC address:

```
# Set MAC address (interface must be down)
ip link set wlan0 down
ip link set wlan0 address 00:11:22:33:44:55
ip link set wlan0 up

# Restore original MAC address (from EEPROM)
# Check /etc/board.conf for original MAC
```

16.5 Hostapd and WPA Supplicant

Control WiFi services for AP and client modes.

16.5.1 Hostapd (Access Point)

```
# Start hostapd for interface
systemctl start hostapd@wlan0.service

# Stop hostapd
systemctl stop hostapd@wlan0.service

# Restart hostapd (apply config changes)
systemctl restart hostapd@wlan0.service

# Check status
systemctl status hostapd@wlan0.service

# Manual hostapd start (debug)
hostapd -d /etc/hostapd/wlan0.conf
```

16.5.2 WPA Supplicant (Client/Station)

```
# Start wpa_supplicant for interface
systemctl start wpa_supplicant@wlan0.service

# Stop wpa_supplicant
systemctl stop wpa_supplicant@wlan0.service

# Restart wpa_supplicant
systemctl restart wpa_supplicant@wlan0.service

# Check status
systemctl status wpa_supplicant@wlan0.service

# Manual wpa_supplicant start (debug)
wpa_supplicant -i wlan0 -c /etc/wpa_supplicant/wlan0.conf -d
```

WPA CLI Interactive Commands:

```
# Connect to wpa_cli interactive mode
wpa_cli -i wlan0

# Common commands:
> status                        # Show connection status
> scan                          # Trigger scan
> scan_results                  # Show scan results
> list_networks                 # Show configured networks
> select_network 0              # Select network by ID
> enable_network 0              # Enable network
> disable_network 0             # Disable network
> reconnect                     # Force reconnection
> reassociate                   # Reassociate with current AP
> disconnect                    # Disconnect from AP
> add_network                   # Add new network
> set_network 0 ssid "NewSSID" # Configure network parameter
> save_config                   # Save configuration to file
> quit                          # Exit wpa_cli
```

16.6 Common Workflows

16.6.1 Create and Configure AP Interface

```
# Create AP interface
iw phy phy0 interface add wlan0 type __ap

# Bring interface up
ip link set wlan0 up

# Start hostapd (configuration in /etc/hostapd/wlan0.conf)
systemctl start hostapd@wlan0.service
```

16.6.2 Create and Configure Station Interface

```
# Create station interface
iw phy phy0 interface add wlan0 type managed

# Bring interface up
ip link set wlan0 up

# Start wpa_supplicant (configuration in /etc/wpa_supplicant/wlan0.conf)
systemctl start wpa_supplicant@wlan0.service
```

16.6.3 Bridge Two WiFi Interfaces

```
# Enable WDS on both interfaces
iw dev wlan0 set 4addr on
iw dev wlan1 set 4addr on

# Create bridge
ip link add name br0 type bridge

# Add interfaces to bridge
ip link set wlan0 master br0
ip link set wlan1 master br0

# Bring up bridge
ip link set br0 up
```

16.7 Troubleshooting

16.7.1 Check Interface Existence

```
# List all network interfaces
ip link show

# Check specific interface
iw dev wlan0 info || echo "Interface does not exist"
```

16.7.2 Verify PHY and Driver

```
# Check WiFi PHY devices
iw list

# Check loaded drivers
lsmod | grep -E "ath11k|ath10k|mac80211"

# Check driver messages
dmesg | grep -E "ath11k|ath10k"
```

16.7.3 Reset Interface

```
# Remove and recreate interface
iw dev wlan0 del
iw phy phy0 interface add wlan0 type __ap
ip link set wlan0 up
```

16.7.4 Debug Logging

Enable Verbose Hostapd Logging:

Run hostapd manually with debug output:

```
systemctl stop hostapd@wlan0
hostapd -i wlan0 /etc/hostapd/wlan0.conf -dd
```

Configure and restart hostapd systemd managed instance:

```
# Edit /etc/hostapd/wlan0.conf
logger_syslog=-1
logger_syslog_level=0            # 0=verbose debug

# Restart and monitor logs
systemctl restart hostapd@wlan0
journalctl -u hostapd@wlan0 -f
```

Enable Verbose WPA Supplicant Logging:

Run wpa_supplicant manually with debug output:

```
systemctl stop wpa_supplicant@wlan0
wpa_supplicant -i wlan0 -c /etc/wpa_supplicant/wlan0.conf -dd
```

Restart wpa_supplcant systemd managed instance:

```
systemctl edit wpa_supplicant@wlan0
# Add: ExecStart=/usr/sbin/wpa_supplicant -c/etc/wpa_supplicant/wlan0.conf -iwlan0 -dd
```

16.7.5 Common Issues and Solutions

Symptom	Possible Cause	Solution
Interface won't come up	Missing firmware	Check `dmesg` for firmware errors, install `linux-firmware` package
No clients connecting	Incorrect security config	Verify WPA settings match between AP and client
Frequent disconnections	Channel congestion	Switch to less congested channel using `iw dev scan`
Poor throughput	Suboptimal channel width	Try different channel widths (20/40/80 MHz)
DFS channel not available	CAC in progress or radar detected	Wait for CAC completion (60s) or select non-DFS channel
Interface stuck in "UNKNOWN" state	Driver or hardware issue	Reload driver: `rmmod ath11k_pci && modprobe ath11k_pci`
Authentication timeout	Weak signal or interference	Check signal strength with `iw dev wlan0 station dump`
Can't set channel	hostapd/wpa_supplicant active	Stop service before manual channel changes

16.8 Related Documentation

- Robonode Hardware - WiFi radio hardware specifications
- Robonode Initialization - Hardware initialization sequence
- WiFi Statistics - WiFi statistics library design
- ATH11K Controls - ATH11K driver controls

17. ATH11K Controls

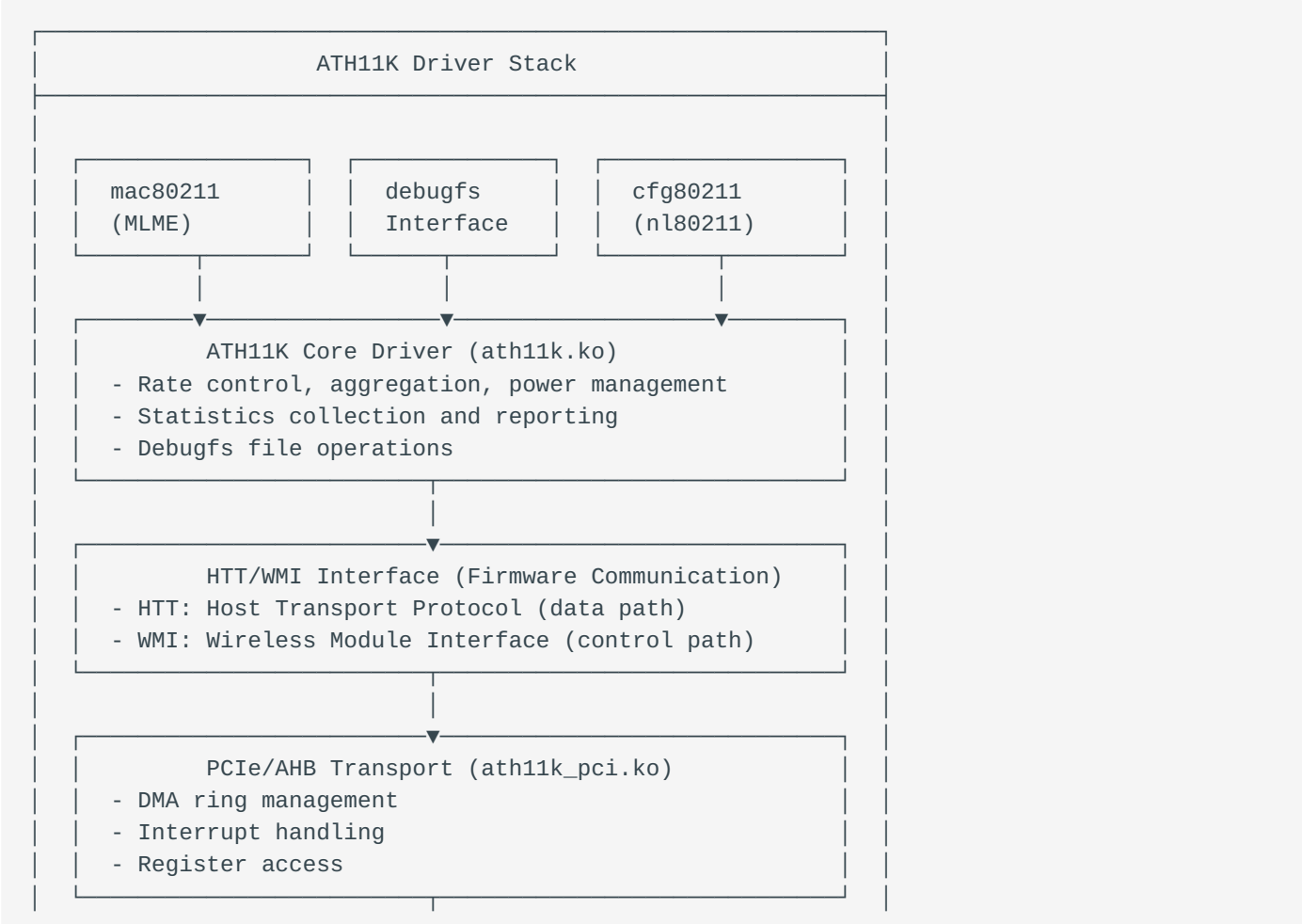
17.1 Overview

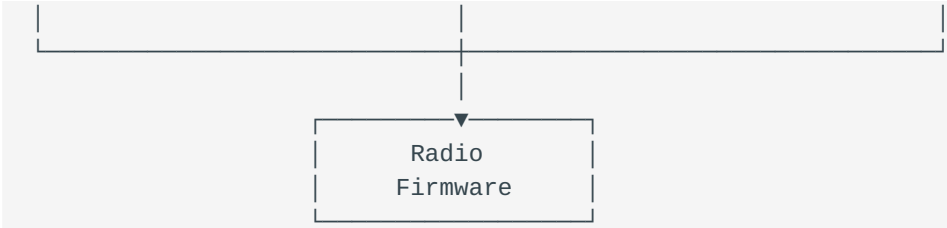
The ath11k driver provides advanced radio control capabilities for monitoring, diagnostics, and configuration. Controls are accessed through the Linux debugfs interface.

Key Capabilities:

- Per-station TX/RX statistics with MCS/bandwidth/NSS details
- Firmware-level device and interface statistics
- Long-distance link optimization (ACK timeout, channel bandwidth)
- Band variant switching (2.4/5/6 GHz)
- Frequency offset tuning
- Debug logging and diagnostics

Driver Architecture:





17.2 debugfs Path Structure

Radio controls are organized under debugfs with the following structure:

Per-Device (SoC):

```
/sys/kernel/debug/ath11k/pci-<bus:device.function>/
```

Per-Radio (MAC):

```
/sys/kernel/debug/ath11k/pci-<bus:device.function>/mac<N>/
```

Per-Station:

```
/sys/kernel/debug/ieee80211/phy<N>/netdev:<ifname>/stations/<MAC>/
```

Per-Interface:

```
/sys/kernel/debug/ieee80211/phy<N>/netdev:<ifname>/
```

Example paths:

- Device: `/sys/kernel/debug/ath11k/pci-0000:01:00.0/`
- Radio: `/sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/`
- Station: `/sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/`
- Interface: `/sys/kernel/debug/ieee80211/phy0/netdev:wlan0/`

To identify radio paths, check `/etc/radios.cfg`:

```
cat /etc/radios.cfg
```

Example output:

```
radio0=/sys/devices/platform/soc@0/10000000.pci/pci0000:00/0000:00:00.0/0000:01:00.0
```

The PCIe bus identifier (`0000:01:00.0`) maps to the debugfs path.

17.3 Feature Controls

17.3.1 ext_tx_stats

Enable per-station TX statistics collection with detailed MCS, bandwidth, NSS, retry counts, and success rates.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/ext_tx_stats`

Input: Hexadecimal filter value (e.g., `0x00000001`)

Output: Current filter value in hex format

```
# Enable
echo "0x00000001" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_tx_stats

# Read status
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_tx_stats
```

17.3.2 ext_rx_stats

Enable per-station RX statistics collection with detailed MCS, bandwidth, NSS, FCS errors, and MPDU counts.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/ext_rx_stats`

Input values:

- `0` - Disable
- `1` - Enable

Output: Current state (0 or 1)

Note: Cannot change while monitor mode is active.

```
# Enable
echo "1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_rx_stats

# Disable
echo "0" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_rx_stats

# Read status
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_rx_stats
```

17.3.3 tx_stats (per-station)

View detailed TX statistics for a connected station. Requires `ext_tx_stats` enabled.

Path: `/sys/kernel/debug/ieee80211/phy<N>/netdev:<ifname>/stations/<MAC>/tx_stats`

Operation: Read-only

Output includes:

- Success/Fail/Retry counts by MCS (HE, VHT, HT, Legacy)
- Bandwidth distribution (20/40/80/160 MHz)
- NSS distribution (1x1, 2x2, 3x3, 4x4)
- Guard Interval distribution
- TX duration
- BA and ACK failures

```
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/tx_stats
```

17.3.4 rx_stats (per-station)

View detailed RX statistics for a connected station. Requires `ext_rx_stats` enabled.

Path: `/sys/kernel/debug/ieee80211/phy<N>/netdev:<ifname>/stations/<MAC>/rx_stats`

Operation: Read-only

Output includes:

- MSDU counts (total, TCP, UDP, AMPDU, non-AMPDU)
- MPDU FCS ok/error counts
- MCS, bandwidth, NSS distribution
- STBC and beamforming counts
- Preamble types
- Reception types (SU/MU-MIMO/MU-OFDMA)
- RU allocation counts
- RX duration

```
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/rx_stats
```

17.3.5 fw_stats/pdev_stats

Physical device statistics from firmware including channel utilization, PHY errors, and TX/RX frame counts.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/fw_stats/pdev_stats`

Operation: Read-only

```
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_stats/pdev_stats
```

17.3.6 fw_stats/vdev_stats

Virtual device statistics for all active interfaces including per-interface TX/RX counts, beacon statistics, and multicast/broadcast counts.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/fw_stats/vdev_stats`

Operation: Read-only

```
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_stats/vdev_stats
```

17.3.7 fw_stats/beacon_stats

Beacon transmission statistics including transmission counts, delivery failures, and DTIM information (AP mode only).

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/fw_stats/beacon_stats`

Operation: Read-only

```
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_stats/beacon_stats
```

17.3.8 soc_dp_stats

SoC-level datapath statistics including RX error ring counts, RXDMA errors, REO errors, TCL ring full counts, and backpressure statistics.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/soc_dp_stats`

Operation: Read-only

```
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/soc_dp_stats
```

17.3.9 bdf_variant

Configure WiFi band variant (2.4 GHz / 5 GHz / 6 GHz) for the radio. This is a module parameter that requires driver reload to take effect.

Path: `/sys/module/ath11k/parameters/bdf_variant`

Input values:

- `2` - 2.4 GHz band
- `5` - 5 GHz band
- `6` - 6 GHz band

Output: Current band variant

Note: Changing band variant is DESTRUCTIVE - requires full driver reinitialization and destroys all interfaces.

```
# Check current band variant
cat /sys/module/ath11k/parameters/bdf_variant

# Set band variant (requires driver reload)
echo "2" > /sys/module/ath11k/parameters/bdf_variant  # 2.4 GHz
echo "5" > /sys/module/ath11k/parameters/bdf_variant  # 5 GHz

# Driver reload required after band change (destroys all interfaces)
rmmod ath11k_pci
modprobe ath11k_pci

# Persistent band setting (recommended method)
echo "options ath11k bdf_variant=5" > /etc/modprobe.d/ath11k_variant.conf
```

Note: Use `radioconf` utility for safe band switching with proper interface management.

17.3.10 ack_timeout

Configure ACK timeout for long-distance links. Timeout compensates for signal propagation delay.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/ack_timeout`

Input range: 1-255 microseconds (recommended: > 40)

Output: Current timeout value

Note: Changes apply instantly, no restart required.

Calculation: Distance (km) × 6.67 µs/km ≈ timeout

Recommended values:

Distance	Timeout
5 km	~50 µs
10 km	~70 µs
15 km	~100 µs
20 km	~135 µs

```
# Set timeout (example for 15km)
echo "100" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ack_timeout
```

```
# Read current value
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ack_timeout
```

17.3.11 chan_bw

Configure narrow channel bandwidth for special applications (half-rate and quarter-rate channels).

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/chan_bw`

Input values:

- `5` - 5 MHz (quarter-rate)
- `10` - 10 MHz (half-rate)
- `20` - 20 MHz (full-rate)
- `0` - Disable custom bandwidth

Output: Current bandwidth in MHz

Note: Changes require VAP restart (hostapd/wpa_supplicant restart).

```
# Set 10 MHz bandwidth
echo "10" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/chan_bw

# Disable custom bandwidth
echo "0" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/chan_bw

# Read current value
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/chan_bw

# Restart services after bandwidth change
systemctl restart hostapd@wlan0.service
systemctl restart wpa_supplicant@wlan0.service
```

17.3.12 cca_threshold

Configure Clear Channel Assessment (CCA) threshold to override the hardware's default carrier sense level. Used to adjust receive sensitivity for environments with high interference or to extend range.

Paths:

- `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/cca_threshold` - Threshold value in dBm
- `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/cca_disable` - Boolean to disable hardware CCA

Input range: -100 to 0 dBm

Usage: Set `cca_disable=1` first to override hardware CCA, then set `cca_threshold` to the desired value.

Configuration: Managed by the WiFi netman handler based on the `cca_threshold` config parameter in roboconf.

```
# Disable hardware CCA (required before setting custom threshold)
echo "1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/cca_disable

# Set CCA threshold to -62 dBm
echo "-62" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/cca_threshold

# Read current values
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/cca_threshold
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/cca_disable
```

Note: Lower threshold values (e.g., -82 dBm) make the radio more sensitive to carrier detection, reducing hidden node issues but potentially decreasing throughput. Higher values (e.g., -40 dBm) make the radio less sensitive, allowing more aggressive transmission but increasing collision risk.

17.3.13 freq_shift

Fine-tune radio operating frequency by specifying offset in Hz from configured channel frequency for intentional frequency adjustment.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/freq_shift`

Input: Frequency offset in Hz (positive or negative)

Output: Current frequency offset in Hz

Note: Changes apply instantly, no restart required.

```
# Shift frequency up by 1 MHz
echo "1000000" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/freq_shift

# Shift frequency down by 1 MHz
echo "-1000000" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/freq_shift

# Read current value
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/freq_shift
```

17.3.14 fw_dbglog_config

Configure firmware debug log level for diagnostic logging.

Path: `/sys/kernel/debug/ath11k/pci-<pci-addr>/mac0/fw_dbglog_config`

Input format: `<param> <value>`

Log levels:

Level	Description
0	VERBOSE (most detailed)
1	INFO
2	INFO_LVL_1
3	INFO_LVL_2
4	WARN
5	ERR (least detailed)

```
# Set log level to INFO
echo "1 1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_dbglog_config
```

17.3.15 htt_peer_stats (per-station)

HTT peer statistics for detailed peer-level diagnostics.

Path: `/sys/kernel/debug/ieee80211/phy<N>/netdev:<ifname>/stations/<MAC>/htt_peer_stats`

Operation: Read-only

```
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/
htt_peer_stats
```

17.4 Monitor Mode Interface

Monitor mode interfaces are used by the spectral scan and WiFi Broadcast services for raw frame access.

Interface creation: The WiFi launcher manages monitor interface lifecycle. When `mon.enabled` is set to `true` in the radio configuration, or when the radio mode is `"WFB"`, the launcher creates a monitor interface (e.g., `wlan0.mon`).

Passive monitor: Created when `mon.enabled: true` with standard radio modes (BSS, NAW, mesh). Used by spectrald for spectral scan data collection.

Active monitor: Created automatically when `radio0.mode: WFB`. The interface has packet injection capability enabled (`active_mon` flag), required for wfb-ng TX operations. Also includes radio parameter configuration (channel, frequency, protocol) applied to the monitor interface.

Configuration:

```
wifi:
  radio0:
    mon:
      enabled: true  # Create passive monitor interface
```

Or for active monitor (automatic in WFB mode):

```
wifi:
  radio0:
    mode: WFB      # Automatically creates active monitor interface
```

Note: `ext_rx_stats` cannot be changed while monitor mode is active.

17.5 Usage Scenarios

17.5.1 Scenario 1: Monitor Station Performance

Objective: Track detailed TX/RX performance for a specific station.

```
# Step 1: Enable extended statistics
echo "0x00000001" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_tx_stats
echo "1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_rx_stats

# Step 2: Identify station MAC address
iw dev wlan0 station dump

# Step 3: View TX statistics
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/tx_stats

# Step 4: View RX statistics
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/rx_stats
```

```
# Step 5: Monitor continuously (refresh every 2 seconds)
watch -n 2 "cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/
tx_stats | head -20"
```

Key metrics to monitor:

- **Success rate:** High success rate indicates good link quality
- **Retry rate:** High retries suggest interference or distance issues
- **MCS distribution:** Higher MCS values indicate better link quality
- **Bandwidth usage:** Shows actual channel width being used
- **BA failures:** Block ACK failures may indicate buffer issues

---

17.5.2 Scenario 2: Debug Connectivity Issues

Objective: Gather diagnostic information for troubleshooting connection problems.

```
# Step 1: Enable firmware debug logging (INFO level)
echo "1 1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_dbglog_config

# Step 2: Enable extended statistics
echo "0x00000001" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_tx_stats
echo "1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_rx_stats

# Step 3: Check firmware stats
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_stats/pdev_stats

# Step 4: Check SoC datapath errors
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/soc_dp_stats

# Step 5: Check per-station statistics
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/tx_stats
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/rx_stats

# Step 6: Monitor system logs
journalctl -f -k | grep ath11k
```

Common issues to look for:

- **High FCS errors in RX stats:** RF interference or signal quality issues
- **High retry counts in TX stats:** Poor link quality, interference, or distance
- **RXDMA errors in soc_dp_stats:** Driver or hardware issues
- **Low MCS rates:** Signal strength or SNR problems
- **BA failures:** Aggregation issues

17.5.3 Scenario 3: Long Distance Link Optimization

Objective: Configure radio for long-distance point-to-point link.

```
# Step 1: Calculate ACK timeout
# Distance in km × 6.67 µs/km ≈ timeout
# Example: 15km × 6.67 = 100µs

# Step 2: Set ACK timeout (example for 15km)
echo "100" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ack_timeout

# Step 3: Verify setting
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ack_timeout

# Step 4: Enable statistics to monitor performance
echo "0x00000001" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_tx_stats
echo "1" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_rx_stats

# Step 5: Monitor retry rates and ACK failures
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/tx_stats |
grep -E "Retry|ACK"

# Step 6: Adjust timeout if needed based on retry rate
# If retries are high, increase timeout
# If retries are low, timeout is adequate
```

Optimization tips:

- Start with calculated timeout and adjust based on observed retry rates
- Monitor link quality over time to detect interference patterns
- Use directional antennas and clear line of sight
- Consider weather conditions that may affect signal propagation

---

17.5.4 Scenario 4: Rate Control Analysis

Objective: Analyze rate adaptation behavior and link quality.

```
# Step 1: Enable extended TX statistics
echo "0x00000001" > /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ext_tx_stats

# Step 2: Generate traffic to specific station
# (use iperf3, ping, or actual application traffic)

# Step 3: Monitor MCS distribution
watch -n 1 "cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/
tx_stats | grep 'MCS\|Success\|Retry'"

# Step 4: Analyze MCS usage over time
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/00:11:22:33:44:55/tx_stats |
grep -A 1 "HE MCS"

# Step 5: Calculate success ratio
# Success / (Success + Fail) ratio should be > 90% for good link
```

Analysis points:

- **MCS distribution:** Should use highest MCS rates for good link quality
- **Success vs Retry ratio:** High retries indicate sub-optimal rate selection or link issues
- **Bandwidth usage:** Verify if using expected channel width (20/40/80/160 MHz)
- **NSS (spatial streams):** Check if using all available antennas

Expected behavior:

- Good link: Predominantly high MCS (MCS 9-11 for HE), low retries
- Marginal link: Mixed MCS rates, moderate retries
- Poor link: Low MCS rates (MCS 0-3), high retries and failures

17.6 Troubleshooting

17.6.1 High Retry or Failure Rates

Symptoms:

- Retry rate >20%
- Failure rate >5%
- Poor throughput

Debugging Steps:

```
# 1. Check link quality
iw dev wlan0 station get <MAC>
# Look for: signal, rx/tx bitrate

# 2. Check channel utilization
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/fw_stats/pdev_stats | grep -A5
"Utilization"

# 3. Check for interference
iw dev wlan0 survey dump
# Look for: channel busy time, noise floor

# 4. Verify ACK timeout (for long distance)
cat /sys/kernel/debug/ath11k/pci-0000:01:00.0/mac0/ack_timeout

# 5. Check rate control behavior
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/<MAC>/tx_stats | grep "HE MCS"
```

Potential Fixes:

Cause	Fix
Long distance	Increase ack_timeout: `echo "200" > .../ack_timeout`
Interference	Change channel: `iw dev wlan0 set channel <N>`
Weak signal	Increase TX power, improve antenna placement
Rate control stuck low	Check signal strength, reduce distance/interference
Channel congestion	Monitor with survey, change to less busy channel

17.6.2 Performance Not Meeting Expectations

Diagnostic Checklist:

```
# 1. Verify PHY capabilities
iw phy phy0 info | grep -E "Band|Frequencies|HE PHY|VHT|HT"

# 2. Check actual vs expected bandwidth
iw dev wlan0 info | grep channel
# Verify HT20/HT40/VHT80/VHT160 etc

# 3. Check NSS (spatial streams)
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/<MAC>/tx_stats | grep NSS
# Verify 2x2, 4x4 etc matching hardware

# 4. Monitor MCS distribution
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/<MAC>/tx_stats | grep "HE MCS
[0-9]\+\|VHT MCS"
# Should use high MCS for good link

# 5. Check aggregation
cat /sys/kernel/debug/ieee80211/phy0/netdev:wlan0/stations/<MAC>/tx_stats | grep "BA\|
AMPDU"
# Verify aggregation is working
```

Optimization Tips:

- Ensure client supports same PHY mode (HE/VHT/HT)
- Use wider channels (80/160 MHz) when possible
- Enable AMPDU aggregation
- Verify antenna configuration (all chains active)
- Check for CPU bottlenecks on host
- Use iperf3 for realistic throughput testing

17.7 Related Documentation

- Robonode Hardware - Radio hardware specifications
- Robonode Initialization - Hardware initialization sequence
- System Initialization - WiFi driver pre-configuration during boot
- WiFi Controls - WiFi radio configuration

17.8 Referenced Code

- **Driver sources:** `drivers/net/wireless/ath/ath11k/`
- **Radio configuration:** `/etc/radios.cfg`
- **Firmware location:** `/lib/firmware/ath11k/<chip>/hw1.0/`
- **Board data files:** `/lib/firmware/ath11k/<chip>/hw1.0/board*.bin`

18. ESConf Usage Guide

18.1 Operation Principles

ESConf manages embedded system configuration through a structured lifecycle approach with centralized state coordination. The system operates in distinct phases, enabling predictable startup sequences and reliable service management.

18.1.1 Core Concepts

- **Phase-Based Lifecycle**: System progresses through boot → launchers → ready phases
- **Launcher-Based Services**: Independent executable modules manage specific system components
- **Centralized State**: Shared state enables coordination between different system services
- **Session Management**: Structured session handling with logging and configuration staging

18.1.2 System Phases

- **Boot Phase**: Initial system setup, configuration loading, state initialization
- **Launcher Phase**: Individual service startup, configuration application, inter-service coordination
- **Ready Phase**: System fully operational, all services running and coordinated

18.2 CLI Reference

18.2.1 Core System Commands

`esconf boot` / `esconf ready`

Core system lifecycle commands for automated system integration (not intended for manual usage).

- `esconf boot` : Initialize system state, load configuration, start session logging, set boot phase
- `esconf ready` : Transition to ready phase, enable launcher triggers, complete boot lifecycle

`esconf status [launcher]`

Display system and launcher status information.

```
# System-wide status
esconf status
```

```
# Specific launcher status
esconf status network
esconf status wifi
```

Shows current system phase, registered launchers, launcher-specific diagnostic information, and configuration summary.

`esconf start [launcher]`

Start system launchers or specific launcher.

```
# Start all registered launchers
esconf start

# Start specific launcher
esconf start network
esconf start wifi
```

Executes launcher start() method, updates registry status, triggers dependent reloads if configured.

`esconf stop [launcher]`

Stop system launchers or specific launcher.

```
# Stop all registered launchers
esconf stop

# Stop specific launcher
esconf stop network
esconf stop wifi
```

Executes launcher stop() method, updates registry status, cleans up launcher state.

`esconf reload [launcher]`

Reload system launchers or specific launcher (stop + start sequence).

```
# Reload all registered launchers
esconf reload

# Reload specific launcher
esconf reload network
esconf reload wifi
```

Useful for applying configuration changes while maintaining state consistency.

18.2.2 Command Options

All launcher commands support direct execution mode:

```
# Skip pre/post processing hooks
esconf start network --direct
esconf stop wifi -D
```

18.3 Boot Lifecycle Management

18.3.1 Session Management

ESConf uses session management to coordinate boot operations and provide predictable configuration:

Session-Based Logging

- Each boot operation creates a unique session ID
- Operations are logged with session context for traceability
- Session data includes timing, status, and error information
- Structured logging helps with debugging and monitoring

Configuration Management

Session management provides staged configuration during boot:

- **Session Configuration**: Staged configuration for predictable boot behavior
- **System Configuration**: `/etc/roboconf/config.yaml` for persistent settings
- **Board Configuration**: On-demand capabilities information loaded by launchers

Configuration sources are prioritized with session configuration taking precedence during boot operations.

18.3.2 Standard Boot Sequence

- **System Initialization**: Session starts, configuration staged
- **Service Startup**: Launchers started with session configuration
- **System Ready**: Session completed, system fully operational

18.4 Launcher Operations

18.4.1 Common Launcher Patterns

Network Launchers

- Configure network interfaces based on zone assignments
- Manage IP addressing (static, DHCP client, DHCP server)
- Coordinate with bridge and routing configuration
- Signal network readiness to dependent services

WiFi Launchers

- Manage wireless interface configuration
- Handle AP mode and client mode operations
- Coordinate with network launchers for IP configuration
- Manage security and authentication settings

Hardware Launchers

- Initialize hardware-specific drivers
- Configure device parameters based on board capabilities
- Provide hardware status and diagnostic information
- Handle hardware-specific error conditions

18.4.2 Dependency Management

Launchers can coordinate through shared state and trigger mechanisms:

- **Shared State**: Launchers communicate readiness and status
- **Trigger Dependencies**: Launchers automatically reload dependents when restarted
- **Configuration Coordination**: Launchers access common configuration sections

18.5 Troubleshooting

18.5.1 Common Issues

Launcher Start Failures

- Check launcher status: `esconf status <launcher>`
- Review configuration for launcher section
- Check system logs for error details
- Verify dependencies are met (other launcher states)
- Restart failed launchers: `esconf start <launcher>`

Configuration Problems

- Validate YAML syntax in configuration files
- Check required configuration sections exist
- Verify file permissions and accessibility
- Review staged vs. persistent configuration sources

State Inconsistencies

- Check system phase: `esconf status`
- Review launcher registry for inconsistent states
- Restart affected launchers: `esconf reload <launcher>`
- Verify system readiness: `esconf status`

18.5.2 Log Analysis

ESConf provides structured logging for debugging:

- **CLI Command Logs**: System command execution and coordination operations
- **Launcher Logs**: Individual launcher operation logs and status updates

Session logs include boot-time operations with session context for traceability and debugging.

This usage guide provides the essential information for operating and troubleshooting ESConf-based embedded systems effectively.

19. NetMan Usage Guide

This guide provides integration information for service integrators incorporating NetMan into their systems.

19.1 Configuration Management

19.1.1 Configuration File Format

NetMan uses a key-value configuration format with hierarchical precedence:

```
# /etc/netman.conf

# Group properties - apply to all interfaces
*.monitoring=enabled
*.mtu=1500

# Pattern-based properties - apply to matching interfaces
eth*.speed=1000
wlan*.power_save=disabled

# Interface-specific properties - highest precedence
eth0.address=192.168.1.100/24
eth0.gateway=192.168.1.1
wlan0.ssid=MyNetwork
```

19.1.2 Configuration Precedence Rules

Configuration resolution follows strict precedence hierarchy:

- **Specific Interface** (`eth0.property=value`) - Highest precedence
- **Pattern Interface** (`eth*.property=value`) - Medium precedence
- **Group Interface** (`*.property=value`) - Lowest precedence

19.1.3 Configuration Loading

Configuration files are loaded in the following order:

- Main configuration file: `/etc/netman.conf`
- Additional files from `/etc/netman.d/` processed alphabetically
- Volatile configuration from `/run/netman/` (runtime configuration)

Later loaded files can override previously set values following the precedence rules. Volatile configuration in `/run/netman/` allows for temporary runtime configuration that doesn't persist across reboots.

19.1.4 Internal Configuration Parameters

NetMan interprets certain configuration parameters internally:

init: Controls automatic interface initialization and dependency handling

```
eth0.init=1          # Emit interface requisite/initialization event when config is
created (even if interface doesn't exist)
eth0.init=eth1       # Emit interface requisite/initialization event when dependent
interface eth1 is created
# eth0.init=          # Interface events only created when interface exists (default
behavior)
```

- When `init=1`: Interface requisite/initialization event is automatically emitted when configuration is created, regardless of whether the interface physically exists
- When `init=<dependent-interface>`: Interface requisite/initialization event is automatically emitted when the specified dependent interface is created
- When `init` parameter is not set: Interface events are only created when the interface physically exists (normal behavior)

19.2 Action Handler Interface

19.2.1 Action Script Execution

NetMan executes a single action handler script located at `/usr/share/netman/netscript.sh` for all events. The script receives comprehensive information about interface state and configuration through environment variables.

19.2.2 Action Handler Environment

The action handler receives two categories of environment variables:

State Parameters

Interface state information provided for all events:

```
NETMAN_IFNAME=eth0                    # Interface name
NETMAN_IFINDEX=2                      # Kernel interface index
NETMAN_EVENT=link-up                  # Current event type
NETMAN_TIMESTAMP=1634567890          # Event timestamp (Unix timestamp)

# Interface properties
NETMAN_MAC=00:11:22:33:44:55         # MAC address (if available)
NETMAN_MTU=1500                       # Maximum transmission unit
```

```
NETMAN_ADMIN_STATE=up                # Administrative state (up/down)
NETMAN_OPER_STATE=up                 # Operational state (up/down/dormant/lowerlayerdown)

# Address information (when available)
NETMAN_IPV4_ADDR=192.168.1.100/24    # Primary IPv4 address with prefix
NETMAN_IPV6_ADDR=2001:db8::1/64      # Primary IPv6 address with prefix
NETMAN_IPV4_ADDRS="192.168.1.100/24 10.0.0.1/8"  # All IPv4 addresses (space-separated)
NETMAN_IPV6_ADDRS="2001:db8::1/64"   # All IPv6 addresses (space-separated)
```

Configuration Parameters

Configuration parameters are passed directly to the action handler using their original parameter names:

Configuration Parameter Propagation:

- Configuration parameter names are passed as-is without prefix or transformation
- Current values are provided with their original parameter name
- Previous values are provided with `OLD_` prefix for change detection

For a configuration parameter `eth0.gateway=192.168.1.1`:

- **Current value**: `gateway=192.168.1.1`
- **Previous value**: `OLD_gateway=192.168.1.1` (when parameter changes)

Example Environment Variables:

```
# Configuration parameters (direct names, no prefix)
gateway=192.168.1.1        # From eth0.gateway=192.168.1.1
dnsserver=8.8.8.8          # From eth0.dnsserver=8.8.8.8
vlanid=100                 # From eth0.vlanid=100
init=1                     # From eth0.init=1

# Previous values (when configuration changes)
OLD_gateway=192.168.1.254  # Previous value if gateway was changed
OLD_dnsserver=8.8.4.4      # Previous value if DNS server was changed
```

19.2.3 Supported Events

NetMan generates the following events that trigger action handler execution:

Interface Events

- **if-create**: Interface was created (physical interface added)
- **if-destroy**: Interface was destroyed (physical interface removed)

- **if-up**: Interface was administratively enabled (ifconfig up)
- **if-down**: Interface was administratively disabled (ifconfig down)
- **if-mac**: Interface MAC address changed
- **if-requisite**: Interface requisition event emitted when interface doesn't exist but is required as requisite for some other interface or config block (triggered by init config parameter)

Link Events

- **link-up**: Physical link became operational (cable connected, wireless associated)
- **link-down**: Physical link became non-operational (cable disconnected, wireless disassociated)
- **link-dormant**: Physical link waiting for external event (e.g., authentication)
- **link-lldown**: Physical link lower layer is down

Address Events

- **ipv4-assigned**: IPv4 address was assigned to interface
- **ipv4-removed**: IPv4 address was removed from interface
- **ipv4-changed**: IPv4 address was modified (prefix length change)
- **ipv6-assigned**: IPv6 address was assigned to interface
- **ipv6-removed**: IPv6 address was removed from interface
- **ipv6-changed**: IPv6 address was modified (prefix length change)

Configuration Events

- **config-changed**: Configuration parameters changed for interface
- **config-created**: New configuration created for interface
- **config-removed**: Configuration removed for interface

19.2.4 Event Timing

From the user perspective, there are two categories of events:

Interface State Events: Originate from actual system state changes reported by the kernel through RTNETLINK or external event injection.

Interface Config Events: Originate from configuration data and its changes when NetMan configuration is reloaded.

Special Case - if-requisite Event: This event is triggered for interfaces with `init=1` or `init=<interface>` configuration, allowing interface requisition even when the interface doesn't physically exist.

19.2.5 Action Handler Development

Error Handling:

- Exit code 0: Success
- Exit code non-zero: Failure (logged by NetMan)
- Use stderr for error messages
- Implement proper cleanup on failure

19.3 Service Operation

19.3.1 Starting and Stopping

```
netmand             # Run in foreground
```

19.3.2 Signal Handling

NetMan responds to the following signals:

Signal	Action	Description
`SIGUSR1`	Configuration reload	Reload all configuration files
`SIGUSR2`	Configuration clear	Clear all configuration
`SIGHUP`	State dump	Dump current state to `/run/netman/netman.dump`
`SIGTERM`	Graceful shutdown	Clean shutdown with resource cleanup

```
# Common signal operations
pkill -USR1 netmand    # Reload configuration
pkill -TERM netmand    # Graceful shutdown
```

19.3.3 Systemd Service Monitoring

View NetMan service logs using journalctl:

```
# View and follow service logs in real-time
journalctl -u netman.service -f
```

19.3.4 Debug Logging

Enable detailed logging for troubleshooting:

```
# Enable debug output (when running manually)
DEBUG=1 netmand

# For systemd service, modify service unit
systemctl edit netman.service
# Add: Environment="DEBUG=1"
systemctl restart netman.service
```

Log Levels:

- **Error**: Critical failures and system errors
- **Warning**: Non-fatal issues and misconfigurations
- **Info**: Service lifecycle and major state changes
- **Debug**: Detailed operation logging (when DEBUG=1)

Debug logging provides:

- Event processing flow
- Configuration parsing details
- Action handler execution and exit codes
- State change tracking
- RTNETLINK message details

19.3.5 State Inspection

Dump current NetMan state for analysis:

```
# Generate state dump
pkill -HUP netmand
```

```
# View state dump
cat /run/netman/netman.dump
```

State dump includes: - All tracked interfaces and their current state - Active configuration parameters per interface - Recent event history - Action handler execution status

19.3.6 Common Troubleshooting

Service Not Starting:

- Check service status: `systemctl status netman.service`
- View recent logs: `journalctl -u netman.service -n 50`
- Verify socket permissions: `ls -l /run/netman/netman.sock`

Configuration Not Applied:

- Verify configuration file syntax
- Check interface name matching (exact vs patterns)
- Review configuration precedence rules
- Reload configuration: `pkill -USR1 netmand`
- View configuration in state dump

Action Handler Issues:

- Verify script exists: `ls -l /usr/share/netman/netscript.sh`
- Check execute permissions: `chmod +x /usr/share/netman/netscript.sh`
- Monitor handler execution: `journalctl -u netman.service -f`
- Review script exit codes in debug logs

Interface Events Not Triggering:

- Verify interface exists: `ip link show`
- Enable debug logging to monitor RTNETLINK
- Review state dump for interface tracking
- Verify configuration matches interface name

19.4 Event Injection Tool

For debugging and testing, NetMan includes the `netevt` tool for manual event injection:

Supported Events:

- **Interface Events**: `if-create`, `if-destroy`, `if-up`, `if-down`, `if-mac`
- **Link Events**: `link-up`, `link-down`, `link-dormant`, `link-lldown`
- **Address Events**: `ip-assign`, `ip-remove`

The tool also supports state initialization mode (`--init`) for bulk interface setup during system initialization.