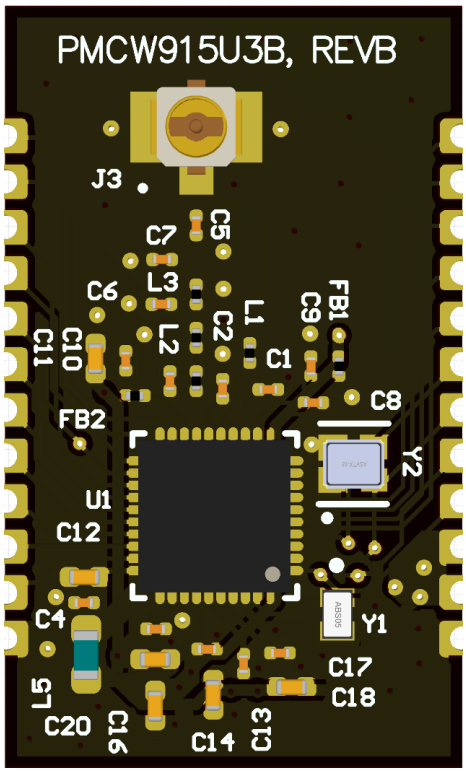
PMCW915U3B

915 MHz Wireless Sensor Gateway Module

UART Host API Reference

UART protocol 1.3.0 · Document D2 · Issue 2.1 · May 2026

Power Micro Controls



> The module is **not yet certified in any country**. The PHY and the enforced envelope are frozen so that what is eventually submitted is what ships. No emissions, band-edge, or PSD measurements exist. This release is for engineering use.

Product: PMCW915U3B Radio Gateway Module **Protocol version:** 1.3.0 **Audience:** host / firmware developers integrating the module

1. Overview

The module is a pre-flashed, source-closed sub-GHz radio gateway. A host MCU (or PC) drives it entirely over a **binary UART protocol** to configure the radio and to retrieve sensor from nodes on a custom 915MHz network. Every module operation is an opcode in this document.

Sensor traffic is **poll/response only**: a sensor node never transmits unless a gateway polls it, and each `SENSOR_*` request produces at most one reply. There is no streaming mode.

All radio internals (the on-air 2-GFSK framing, addressing, and byte offsets) are hidden behind this API. The RF envelope (channel plan 0..25 and the +20.0 dBm maximum TX power) is enforced in firmware; values outside it are rejected, never applied. The frequency-hopping parameters in D3 persist but the hop engine is not active in this release, so the module operates on `operating_channel`.

2. Physical transport

The module exposes two UARTs. **COM1 is the machine API** — the framed binary protocol this document defines, and the only surface host software may integrate against. **COM0 is a human-friendly text console** (secondary command surface): the same operations — config get/set/commit, per-EUI sensor status and block/page reads — as line-oriented text commands for bench and PC use (carrier boards commonly bridge it to USB through a CP2105). The text grammar is documented separately and may evolve without a `protocol_version` bump; do not build machine integrations on it.

COM0 also accepts framed binary: a line beginning with the `0xA5` sync byte is demultiplexed to the same dispatcher COM1 uses. That path exists for bench tooling only — a binary client on COM0 additionally receives the console's ASCII output interleaved with its responses, so production hosts must use COM1.

Role	Name	Peripheral	TX	RX	Baud	Framing
Public API	COM1	EUSART1	PA05	PA06	460800	8N1
Text console (secondary)	COM0	EUSART0	PA07	PA08	460800	8N1

Both run **8N1, no flow control**. Wire the host TX→module RX, host RX→module TX, and share ground. The module is the DCE-style peer: it answers requests and emits asynchronous indications.

3. Frame format

Every message — in both directions — is a single framed packet:

Field	Size (bytes)	Notes
SOF	2	0xA5 0x5A fixed sync
LEN	2	u16 LE. Length of (OPCODE..PAYLOAD), i.e. 1 + payload length. Excludes SOF, SEQ and CRC.
SEQ	1	Host-chosen tag, echoed in the matching response. 0x00 for async indications. Not counted by LEN.
OPCODE	1	See opcodes[]
PAYLOAD	LEN-1	Opcode-specific
CRC16	2	CRC-16/CCITT-FALSE (poly 0x1021, init 0xFFFF) over LEN..PAYLOAD, u16 LE

- **Byte order** for multi-byte protocol fields (LEN, u16/u32 payload values) is **little-endian**, unless a payload field is explicitly ASCII or big-endian (the sensor record timestamp is big-endian; see D7).
- **CRC** is **CRC-16/CCITT-FALSE** (poly `0x1021`, init `0xFFFF`, no reflection, no final XOR), computed over `LEN..PAYLOAD` (everything between SOF and CRC16) and transmitted little-endian.
- **`LEN`** counts `OPCODE` + `PAYLOAD` only (it excludes SOF, LEN itself, and CRC). Minimum `LEN` is 1 (opcode, empty payload).
- Receivers must bound their parse buffer at **`SILADS_FRAME_MAX_BYTES` (2200)**; a decoded sensor page is the largest payload.

Request / response model

- The host sets **`SEQ`** to any value; the module echoes the same `SEQ` in the response so concurrent requests can be matched.
- Module-originated **asynchronous indications** (`SENSOR_DATA_IND`, `SENSOR_STATUS_IND`, `SENSOR_NO_RESPONSE_IND`, and `ACK`/`NACK` for fire-and- forget requests) use **`SEQ = 0x00`**.
- A malformed or unsupported request is answered with **`NACK`** carrying a status code from §5.

4. Opcodes

Code	Name	Dir	Kind	Purpose
0x01	`MOD_INFO`	H->M	req/resp	Identify the module.
0x02	`MOD_STATUS`	H->M	req/resp	Live module status.
0x10	`CFG_GET`	H->M	req/resp	Read one configuration parameter.
0x11	`CFG_SET`	H->M	req/resp	Set one parameter in the RAM working copy (validated, not yet persisted).
0x12	`CFG_COMMIT`	H->M	req/resp	Validate + persist the working copy to NVM3 + apply live.
0x13	`CFG_FACTORY_RESET`	H->M	req/resp	Load compile-time defaults, then commit.
0x14	`CFG_RELOAD`	H->M	req/resp	Discard uncommitted RAM edits; reload from NVM3.
0x20	`SENSOR_READ_LATEST`	H->M	req/resp	Request the most-recent event page from a target sensor.
0x21	`SENSOR_READ_BLOCK`	H->M	req/resp	Request a specific block/page from a target sensor.
0x22	`SENSOR_STATUS`	H->M	req/resp	Request status (battery/RSSI/block counter) from a target sensor.
0x23	`SENSOR_FLASH_RESET`	H->M	req/resp	Reset a target sensor's flash event counters.
0x30	`SENSOR_DATA_IND`	M->H	async	A sensor data page, decoded or raw.
0x31	`SENSOR_STATUS_IND`	M->H	async	A sensor status reply.
0x32	`SENSOR_NO_RESPONSE_IND`	M->H	async	A target sensor did not answer within the retry budget.
0x40	`FW_UPDATE_ENTER`	H->M	req/resp	Key-gated reboot into the UART XMODEM bootloader (Phase 3).
0x50	`GPIO_CONFIG`	H->M	req/resp	Configure one general-purpose pin (mode + initial output level).
0x51	`GPIO_WRITE`	H->M	req/resp	Drive one pin configured as an output.
0x52	`GPIO_READ`	H->M	req/resp	Read the level of one configured pin.
0x53	`GPIO_WRITE_MASK`	H->M	req/resp	Atomically drive several output pins in one operation.
0x54	`GPIO_READ_ALL`	H->M	req/resp	Read all 12 pins at once.
0x55	`GPIO_CHANGE_IND`	M->H	async	An input pin selected by gpio_irq_mask changed level.
0x7E	`ACK`	M->H	async	Generic positive acknowledgement.
0x7F	`NACK`	M->H	async	Generic negative acknowledgement with an error code.

4.1 Payload detail (per opcode)

The generator emits opcode names and directions; the payload shapes below are the contract each opcode carries. Types: `u8/u16/u32` little-endian, `i16` signed little-endian, `ascii[N]` fixed-width ASCII (space/NUL padded).

- **`MOD_INFO` (0x01)** → response: `banner ascii[48]`, `protocol_version u8[3]` (major, minor, patch), `fw_version u8[4]`, `local_eui ascii[16]`, `phy_id ascii[40]` (NUL-padded), `channel_max u8`, `tx_power_max_ddbm i16`. Use this first to confirm the module's protocol version matches your host build.
- **`MOD_STATUS` (0x02)** → response: `rf_state u8`, `last_rssi_dbm i16`, `last_error u16`, `boot_count u32`, `hop_enabled u8`, `operating_channel u8`.
- **`CFG_GET` (0x10)** → request: `param_id u16`. Response: `param_id u16`, `type u8`, `value[]` (width per parameter — see D3). (Reserved: an indexed param would take an extra `index u8` after `param_id`; as of 1.3.0 no parameter is indexed — array parameters such as `hop_table` and `gpio_mode` are transferred whole, one element per byte.)
- **`CFG_SET` (0x11)** → request: `param_id u16` [`index u8` if indexed] `value[]`. Response: `status u16`. The value is validated and staged in RAM; it is **not persisted** until `CFG_COMMIT`. Out-of-range → `ERR_RANGE`; read-only → `ERR_READONLY`; beyond the frozen RF envelope → `ERR_CERT_LIMIT`.

- **`CFG_COMMIT` (0x12)** → response: `status u16`. Validates the whole working set, writes it atomically to NVM3, and applies it live (retunes channel/power, updates timeouts and the hop table). Survives reboot.
- **`CFG_FACTORY_RESET` (0x13)** / **`CFG_RELOAD` (0x14)** → response: `status u16`. Reset-to-defaults-then-commit, and discard-uncommitted-edits.
- **`SENSOR_READ_LATEST` (0x20)** → request: `eui ascii[16]` (the sensor's 64-bit unique ID as 16 ASCII-hex chars), `page u8`. Immediate `ACK`; the page arrives later as `SENSOR_DATA_IND`. A malformed EUI → `NACK ERR_TARGET`.
- **`SENSOR_READ_BLOCK` (0x21)** → request: `eui ascii[16]`, `block u16`, `page u8`. Immediate `ACK`; page arrives as `SENSOR_DATA_IND`.
- **`SENSOR_STATUS` (0x22)** → request: `eui ascii[16]`. Immediate `ACK`; reply arrives as `SENSOR_STATUS_IND`.
- **`SENSOR_FLASH_RESET` (0x23)** → request: `eui ascii[16]`, `ack_flag u8`. Response: `status u16`.

The module is **stateless about device identity**: it holds no roster and any well-formed EUI may be polled. Enrollment/pairing, poll prioritization, and duplicate suppression are host-side (GD32/AWS) responsibilities. With `rx_forward_enable=1` the module also emits indications for sensor frames it hears in response to **other gateways'** polls on the same channel — overlapping gateways all report what they receive, and the backend dedups (this is the multi-gateway redundancy model).

Retrieval is **event-indexed, not time-ranged**. One sensor event is one flash block of **64 pages**, and each page holds 45 records (see D7). Sensor nodes have no real-time clock — record timestamps are free-running ticks — so a host maps wall-clock time onto an event index using its own heartbeat observations rather than asking the module for a time range. Budget accordingly: one poll transaction (command plus reply) takes about **0.7 s**, and a full 64-page event retrieval about **45–50 s**. That time is shared site-wide by every module within radio range, since the channel carries roughly 1.3–1.4 transactions per second in total.

- **`SENSOR_DATA_IND` (0x30)** ← async: `target_eui ascii[16]`, `block u16`, `page u8`, `record_count u8`, `format u8` (`0`=raw 2048-byte page, `1`=decoded records), then the payload. `format` follows the `decode_records` config param. For `format=1` each record is **53 bytes, packed little-endian**, fields in `silads_sensor_record_t` declaration order, and a full 45-record page arrives as **multiple consecutive indications** (≤ 23 records each, same `eui`/`block`/`page`) because 45 records would exceed `SILADS_FRAME_MAX_BYTES`; hosts reassemble in arrival order.
- **`SENSOR_STATUS_IND` (0x31)** ← async: `target_eui ascii[16]`, `battery_mv u16`, `rssi_dbm i16`, `block_counter u16`.
- **`SENSOR_NO_RESPONSE_IND` (0x32)** ← async: `target_eui ascii[16]`. Emitted after `no_response_retries` unanswered transmissions to that target.
- **`FW_UPDATE_ENTER` (0x40)** → request: `unlock_key u32` (`0x0B00710E`). Response: `status u16`. **Not implemented in this release.** The opcode and its key gate exist and the frame is parsed, but there is no bootloader yet: the module answers `NACK ERR_STATE` to every `FW_UPDATE_ENTER`, correct key or not, and never reboots. Field update is a later phase.

GPIO opcodes (0x50–0x55)

The module exposes **twelve** general-purpose pins on its headers. They are addressed by **logical index 0..11 only** — never by SoC port/pin name — so the host is decoupled from the module's pinout:

Index	SoC pin	Header	Index	SoC pin	Header
0	PB01	H1-1	6	PC02	H2-7
1	PB00	H1-2	7	PC03	H2-6
2	PA00	H1-3	8	PC04	H2-5
3	PA04	H1-7	9	PC05	H2-4
4	PC00	H2-9	10	PC06	H2-3
5	PC01	H2-8	11	PC07	H2-2

This mapping is the pin-safety property of the API: the pins the module owns — the two UARTs (PA05–PA08), the debug interface (PA01–PA03) and the PA/LNA control lines (PD02–PD03) — have **no logical index**, so no host command can name them and no host mistake can reconfigure them. An index outside 0..11 is rejected with `ERR_RANGE`.

Pin modes (used by `GPIO_CONFIG` and by the `gpio_mode` parameter): `0` = disabled / high-impedance (factory default), `1` = input, `2` = input with pull-up, `3` = input with pull-down, `4` = output push-pull, `5` = output open-drain.

`GPIO_CONFIG`, `GPIO_WRITE` and `GPIO_WRITE_MASK` take effect **immediately** and are **not persisted**; the power-on state of the pins comes from the `gpio_mode`, `gpio_out_default` and `gpio_irq_mask` parameters (D3), which are staged with `CFG_SET` and written with `CFG_COMMIT` like any other parameter. Where these opcodes are documented as returning `status u16`, the module answers `ACK` on success and `NACK` carrying the error code otherwise.

- **`GPIO_CONFIG` (0x50)** → request: `pin u8` (0..11), `mode u8` (0..5), `initial u8` (0/1, applied only when `mode` is an output). Response: `status u16`. Out-of-range pin, mode, or initial level → `ERR_RANGE`. Reconfiguring a watched input resynchronizes the change detector, so the reconfiguration itself never produces a `GPIO_CHANGE_IND`.
- **`GPIO_WRITE` (0x51)** → request: `pin u8`, `value u8` (0/1). Response: `status u16`. The pin must currently be in an output mode, otherwise `ERR_PIN_MODE`.
- **`GPIO_READ` (0x52)** → request: `pin u8`. Response: `pin u8`, `value u8`. Reads the live level of an input or the driven level of an output. A pin left in mode 0 (high-impedance) has no meaningful level and is rejected with `ERR_PIN_MODE`.
- **`GPIO_WRITE_MASK` (0x53)** → request: `mask u16` (bit *n* selects pin *n*), `values u16` (bit *n* is the level for pin *n*). Response: `status u16`. Bits above 11 address nothing and make the whole request `ERR_RANGE`. The selection is validated in full before anything moves: if any selected pin is not an output the request fails with `ERR_PIN_MODE` and **no** pin changes. Selected pins that share a port switch on the same cycle; pins outside the mask are untouched.
- **`GPIO_READ_ALL` (0x54)** → request: empty. Response: `values u16`, `configured_mask u16`. Bit *n* of `configured_mask` is set when pin *n* is in some mode other than 0; level bits for unconfigured pins read 0 and are meaningless by contract.
- **`GPIO_CHANGE_IND` (0x55)** ← async: `pin u8`, `value u8`, `timestamp u32` (module BURTC tick, 32768 Hz). Emitted when a pin selected by `gpio_irq_mask` and configured as an input changes level; mask bits for output or disabled pins are ignored. Edge detection is **polled** in the module's main loop, not interrupt-driven — the EFR32 series-2 external-interrupt channels are grouped by pin number and could not cover all twelve pins without silently dropping edges. A pulse narrower than one loop iteration can therefore be missed; debounce and stretch short pulses on the carrier board.
- **`ACK` (0x7E) / `NACK` (0x7F)** ← `seq u8`, `opcode u8` [`, error_code u16`].

5. Status / error codes

Returned in `NACK` and in every `status u16` response field.

Code	Symbol	Meaning
0x0000	`SILADS_MOD_OK`	Success
0x0001	`SILADS_ERR_CRC`	Frame CRC mismatch
0x0002	`SILADS_ERR_LENGTH`	LEN inconsistent or buffer overrun
0x0003	`SILADS_ERR_OPCODE`	Unknown opcode
0x0010	`SILADS_ERR_PARAM_ID`	Unknown configuration parameter id
0x0011	`SILADS_ERR_RANGE`	Value outside the allowed range
0x0012	`SILADS_ERR_READONLY`	Parameter is read-only
0x0013	`SILADS_ERR_CERT_LIMIT`	Rejected: would exceed the certified RF envelope
0x0020	`SILADS_ERR_NVM`	NVM3 read/write failure
0x0030	`SILADS_ERR_TARGET`	Malformed sensor EUI (must be 16 ASCII-hex chars)
0x0031	`SILADS_ERR_BUSY`	A radio request is already outstanding
0x0032	`SILADS_ERR_NO_RESPONSE`	Sensor did not answer within retries
0x0040	`SILADS_ERR_STATE`	Operation invalid in the current RF state
0x0050	`SILADS_ERR_PIN_MODE`	GPIO pin is not configured for the requested operation

6. Worked example

Set the operating channel to 12 and persist it:

```
# CFG_SET operating_channel(0x0002) = 12
host -> A5 5A 04 00 2A 11 02 00 0C <crc16_lo> <crc16_hi>
        |SOF | LEN | S| OP|par_id| v |
   LEN = 4  (OPCODE 0x11 + payload: param_id u16 + value u8 = 4 bytes)
   SEQ = 0x2A
module -> A5 5A 03 00 2A 7E 2A 11 <crc>      # ACK (payload: seq 0x2A, opcode 0x11)

# CFG_COMMIT
host -> A5 5A 01 00 2B 12 <crc>
module -> A5 5A 03 00 2B 7E 2B 12 <crc>      # ACK -- committed + applied
```

Rejecting an out-of-envelope value (channel 30 > channel_max 25):

```
host -> A5 5A 04 00 2C 11 02 00 1E <crc>        # CFG_SET channel = 30
module -> A5 5A 05 00 2C 7F 2C 11 13 00 <crc>   # NACK (seq 0x2C, opcode 0x11,
                                                #       error ERR_CERT_LIMIT 0x0013 LE)
```

7. Versioning

`MOD_INFO` reports `SILADS_PROTOCOL_VERSION_STR`. Bump the **major** on any breaking change to frames/opcodes/params/records; **minor** for additive, backward-compatible additions. A host should refuse a module whose **major** differs from the `host_protocol.h` it was built against.