PDF 4.3 – Message Formats & Contracts

1. Purpose

This document defines the **message formats and contracts** used for communication between the MCU and the Linux processor.

It establishes: - how messages are structured - what guarantees are (and are not) provided - how messages can evolve safely over time

This document builds directly on the validated bridge demonstrated in **PDF 4.2 / 4.2.1**.

---

2. Why Message Contracts Matter

Once inter-processor communication is available, uncontrolled message growth quickly becomes a source of bugs.

Message contracts: - make assumptions explicit - reduce ambiguity during debugging - allow independent evolution of MCU and Linux software

In this project, message contracts are treated as **first-class system artifacts**, not ad-hoc implementation details.

---

3. Design Principles

All Phase 4 message formats follow these principles:

- **Explicit structure** – every field has a defined meaning
- **Versioned contracts** – formats can evolve without breaking compatibility
- **Small payloads** – minimize bridge load and latency
- **Fail-safe behavior** – malformed or unexpected messages are ignored, not fatal

---

4. Message Direction and Scope

For Phase 4:

- Direction: **MCU → Linux only**
- Purpose: transport raw data and system events
- No control or configuration messages are sent from Linux to MCU

Bidirectional messaging may be introduced in later phases if required.

---

5. Message Categories

Messages are grouped into logical categories:

1. **System / Heartbeat messages**
2. Used to confirm connectivity and liveness
3. Example: `bridge_alive`
4. **Data messages**
5. Transport raw CAN-related information
6. Introduced incrementally in later Phase 4 steps
7. **Diagnostic messages**
8. Report errors, drops, or abnormal conditions
9. Non-fatal and informational

---

6. Versioning Strategy

Every message payload includes a **version field**.

Rules: - Version numbers are integers - Increment version only when the payload structure changes - Linux must tolerate older versions - MCU never attempts to parse or adapt based on version

This ensures backward compatibility during development.

---

7. Baseline Message Contract (v0)

The baseline contract, validated in Phase 4.2, is intentionally minimal.

Message name

- `bridge_alive`

Payload (v0)

Field	Type	Description
version	uint8	Message format version
counter	uint32	Incrementing heartbeat value

This payload is sufficient to validate: - ordering - delivery reliability - message pacing

---

8. Planned CAN Frame Message Contract (Preview)

A future message contract (introduced incrementally) will transport raw CAN frames.

Planned fields:

Field	Type	Description
version	uint8	Message format version
timestamp	uint32	MCU reception time (ms)
can_id	uint32	Arbitration ID
dlc	uint8	Data length code
data[8]	uint8[8]	Raw CAN payload
flags	uint8	Standard/extended, error indicators

This contract is documented here but **not yet implemented**.

---

9. Message Size Constraints

To protect the MCU and bridge:

- Messages are kept small and bounded
- One CAN frame per message in early stages
- No dynamic allocation based on message size

Batching may be introduced later with explicit limits.

---

10. Error Handling Rules

If a message: - has an unknown name - has an unsupported version - has missing or malformed fields

Then: - Linux ignores the message - A diagnostic log may be generated - MCU behavior is unaffected

No retries or acknowledgements are implemented in Phase 4.

---

11. Relationship to Buffering

Message contracts are independent of MCU buffering:

- The MCU buffer contains raw frames
- Message contracts define how data is serialized across the bridge

This separation allows buffering strategy to evolve without changing message definitions.

---

12. Implementation Reference

Reference implementations for message contracts will appear in:

- `MCU-Phase4.x.ino`
- Linux Phase 4 application modules

Source code is intentionally excluded from this document.

---

13. What This Document Does Not Cover

This document does not yet define:

- throughput limits
- batching strategies
- persistence formats (MF4)
- predictive maintenance logic

Those topics are introduced in later phases.

---

14. Next Document

Proceed to **PDF 4.4 – First CAN Frame Across the Bridge**.