📄 PDF 6.1 — Why MF4?

---

1. Scope of This Document

This document explains **why the MF4 file format is selected** as the primary logging format for Phase 6 of the project.

It does **not** describe how to write MF4 files or replay them (that is Phase 6.2 and 6.3).
It does **not** describe system rules or predictive models (those are Phase 7).

---

2. Phase Context

At the end of Phase 5, the system achieved:

- Real CAN frames received from multiple ECUs
- Decoding on the Linux side using DBC
- Stable throughput at automotive-relevant rates
- Signal-level values verified in real time

Phase 6 addresses a new requirement:

Persist decoded data + raw frame context for replay and predictive maintenance

This introduces three new constraints:

1. **Storage efficiency** (flash is finite)
2. **Replay fidelity** (time alignment matters)
3. **Standardization for tooling** (offline debug, ML, validation)

3. Logging Requirements

The logging subsystem must support the following high-level goals:

3.1 Technical Goals

Requirement	Reason
Timestamp accuracy	Required for replay + rules
Multi-ECU frame handling	CAN bus is multiplexed
Binary encoding	Text is too large
Low CPU overhead	Linux MPU is resource-constrained
High write throughput	Automotive CAN saturates easily
Chunking	Prevents unbounded file growth
Efficient replay	Used for post-event analysis

3.2 Predictive Maintenance Goals

Predictive logic requires:

- historical context (trend windows)
- cross-sensor correlation
- anomaly detection with time alignment
- replay for diagnosis

Example for this project:

degradation in harness voltage is detectable only when comparing **ECU_A**, **ECU_B**, and **DCDC** over time

Without time-aligned logging, this cannot be validated.

---

4. Evaluated Storage Formats

The following formats were considered:

Format	Verdict	Reason
CSV	❌ Reject	Too large, no timestamps, no binary, no replay
JSON / NDJSON	❌ Reject	CPU heavy, huge size, not replayable
SQLite	⚠ Conditional	Good indexing but slow writes
Proprietary binary	⚠ Custom	Flexible but no ecosystem support
MDF / MF4	✅ Preferred	Industry standard + efficient + replay
ROS Bag	❌ Reject	Overkill + robotics stack baggage
Parquet	❌ Reject	Columnar offline storage only

MF4 was the only option meeting both **data** and **automotive** constraints.

---

5. What MF4 Provides

MF4 (ASAM MDFv4) is widely used for:

- automotive sensor logging
- dyno testing
- ECU validation
- powertrain calibration
- ADAS data pipelines

Capabilities relevant to our platform:

5.1 Replay / Time Basis

MF4 supports:

- absolute timestamps
- relative timestamps
- time synchronization
- monotonic sequences

5.2 Binary Efficiency

Typical reduction vs CSV:

- **3× to 20× smaller**
- **zero float parsing**
- **no JSON overhead**

5.3 Structure

MF4 handles:

- raw CAN frames
- decoded channels
- metadata
- sample rates
- ECU identifiers
- DBC context (optional)

These align with CAN testing workflows.

---

6. Why MF4 Works for Predictive Maintenance

The predictive use case requires:

- trend windows (e.g., last 30s of ECU_A voltage)
- cross-signal logic (compare ECU_A vs ECU_B vs DCDC)
- event replay (before/after anomaly)
- post-processing on desktop tools

MF4 enables:

Phase	Benefit
Edge	Efficient write + timestamp

Offline	High-fidelity replay
Training	Statistical extraction
Validation	Compare predicted vs actual degradation

This is the same workflow used in:

- fleet analytics
- autonomous development
- powertrain calibration
- telematics data backhaul

---

7. Fleet / Telematics Analogy

Traditional telematics:

capture → compress → upload → analyze → alert

Our architecture is similar but **inverts the intelligence**:

capture → decode → evaluate rules → log selectively → alert

MF4 fits both modes.

---

8. Storage Strategy for This Project

For Phase 6, we define:

Chunk size:
2 minutes per file

Content per chunk:

- raw CAN frames
- decoded signals
- timestamps
- metadata
- rule state (optional, Phase 7)

Retention policy:
(To be decided in Phase 8)

---

9. Why Not Delete Good Chunks?

As discussed earlier:

Deleting “good” data loses:

- trend context
- baseline learning
- degradation slope
- correlation history

Baseline is important for predictive systems.
Therefore deletion is deferred to Phase 8 (policy).

---

10. Next Phase After This

PDF 6.2 will cover:

How to write MF4 on the Linux MPU

With:

- Python tooling
- buffering strategy
- CPU profiling
- size estimates
- throughput targets

---

11. Conclusion

MF4 is chosen because it satisfies:

- technical constraints
- predictive maintenance needs
- automotive standards
- replay and debugging workflows
- storage efficiency

This format enables Phase 7 predictive logic and Phase 8 alerts.