CAN Fundamentals (IDs, DLC, Frames)

1. Purpose of This Document

This document introduces the **fundamental concepts of CAN (Controller Area Network)** required to understand, validate, and debug CAN traffic before any microcontroller software is written.

Phase 2 deliberately starts **without hardware-specific or software-specific implementation**, focusing instead on protocol-level understanding. This ensures that later design and implementation steps are grounded in correct mental models of how CAN actually works.

---

2. Why CAN Fundamentals Matter

Many CAN-based projects fail or behave unpredictably not because of software bugs, but because of misunderstandings of: - CAN identifiers - Frame structure - Bus arbitration - Payload length and timing

By establishing a shared understanding of these fundamentals early, we reduce ambiguity and make validation possible at every later phase.

---

3. What Is CAN?

CAN (Controller Area Network) is a **message-based, broadcast communication protocol** widely used in automotive and industrial systems.

Key characteristics: - No source or destination addresses - Messages are identified by **IDs**, not node addresses - All nodes see all messages - Arbitration is handled on the bus

This design supports robustness and deterministic communication under contention.

---

4. CAN Frame Types (Overview)

This project focuses on **Classical CAN data frames**, which are sufficient for learning, validation, and monitoring purposes.

The main frame types are: - Data frames (used to carry application data) - Remote frames (request data; not used in this project) - Error frames (generated automatically by controllers) - Overload frames (rarely encountered)

Only **data frames** are considered in the initial phases of this project.

5. CAN Identifiers (IDs)

5.1 Purpose of the CAN ID

The CAN ID: - Identifies the *meaning* and *priority* of a message - Is used for bus arbitration - Does **not** identify a sender or receiver

Lower numerical IDs have **higher priority** on the bus.

5.2 Standard vs Extended IDs

- **Standard ID**: 11-bit identifier (0x000–0x7FF)
- **Extended ID**: 29-bit identifier (0x00000000–0x1FFFFFFF)

This project supports both, but examples initially focus on **standard IDs** for simplicity.

---

6. Data Length Code (DLC)

The Data Length Code (DLC): - Indicates how many data bytes are present in the frame - Ranges from 0 to 8 bytes for Classical CAN

Important notes: - DLC describes payload length, not message meaning - A DLC of 8 does not imply 8 valid signals

Understanding DLC is critical when validating received frames.

---

7. CAN Frame Payload

The CAN payload: - Contains raw data bytes - Has no inherent meaning without external interpretation - Is interpreted using specifications such as **DBC files** (introduced later)

At this stage, payloads are treated as **opaque byte arrays**.

---

8. Arbitration and Bus Access (Conceptual)

When multiple nodes transmit simultaneously: - Arbitration occurs bit-by-bit on the ID field - Dominant bits override recessive bits - The message with the lowest ID wins

Losing nodes automatically retry without data corruption.

This mechanism ensures deterministic priority handling.

9. Relating CAN Frame Fields to the Physical Bus

To connect the abstract CAN concepts discussed so far with what actually appears on the wire, this section references an illustrative CAN frame and physical-layer waveform.

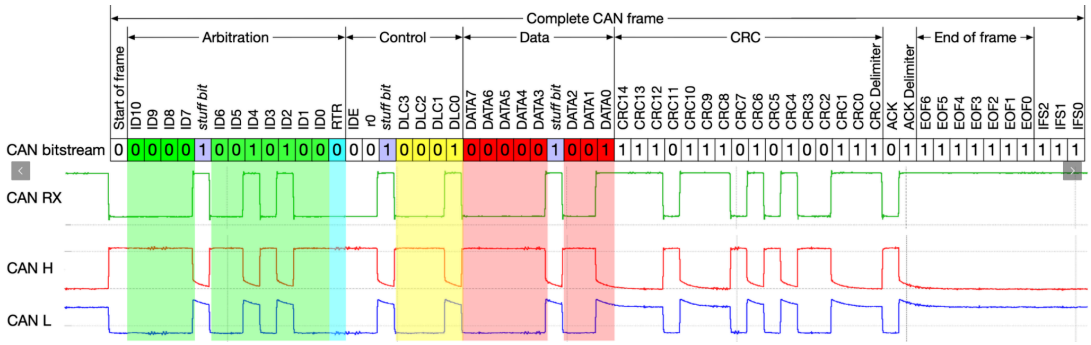

Image source: CAN bus article on Wikipedia (https://en.wikipedia.org/wiki/CAN_bus)

The referenced image shows a single Classical CAN data frame represented at three levels:

1. **Logical CAN frame fields** (top)
2. **Bitstream as seen by the receiver** (CAN RX)
3. **Physical differential signals** (CAN_H and CAN_L)

9.1 Logical CAN Frame Fields

At the highest level, a CAN data frame is divided into well-defined fields:

- **Arbitration field**: Contains the CAN Identifier (ID) and determines message priority on the bus
- **Control field**: Contains the Data Length Code (DLC)
- **Data field**: Contains the payload bytes (0–8 bytes for Classical CAN)
- **CRC field**: Used by CAN controllers to detect transmission errors
- **End of Frame**: Marks the completion of the message

At the software level, applications typically interact only with the **ID, DLC, and Data** fields. All other fields are handled automatically by the CAN controller hardware.

9.2 CAN RX Bitstream

The CAN RX trace represents the logical bitstream reconstructed by the receiver.

Key points: - Bits are transmitted serially - CAN uses two logical states: - **Dominant (0)** - **Recessive (1)** - Arbitration occurs directly on these bits during transmission

Although most software developers never see this bitstream, it is what the CAN controller actually decodes internally.

9.3 Physical Layer: CAN_H and CAN_L

The lower traces show the physical CAN signals:

- **CAN_H** (high)
- **CAN_L** (low)

CAN uses **differential signaling**: - For a **dominant bit**, CAN_H rises and CAN_L falls - For a **recessive bit**, both lines move toward the same voltage level

The receiver detects bits based on the **voltage difference** between CAN_H and CAN_L, not their absolute values. This makes CAN robust against electrical noise and interference.

Understanding this physical behavior is important when diagnosing: - Wiring issues - Termination problems - Signal integrity faults

---

10. Validation Perspective

From a validation standpoint, each received CAN frame can be checked for: - Valid ID range - Expected DLC - Frame arrival rate - Absence of unexpected IDs

These checks will form the basis of later test strategies.

---

11. What This Document Does Not Cover

This document intentionally does **not** cover: - Bit timing and sampling points - CAN FD specifics - Electrical signaling details - Software drivers or APIs

These topics are introduced only when needed.

---

11. Next Steps

With CAN fundamentals established, the next document will cover:

- **PDF 2.2 – CAN Transmit Setup**
    - How test frames are generated
    - How traffic is controlled and observed
    - Preparing for validation against real hardware

This completes the conceptual foundation for Phase 2.