Nova-CAN Communication Standard

Taaj Street, Matt van Wijk, Anthony Verbini, Harrison Ryan

May, 2025

Contents

1 Introduction

This specification is heavily inspired by the OpenCyphal protocol specification, specifically Cyphal/CAN. There are changes to make the protocol simpler and more applicable to NovaRover's use case.

2 Communication Model

Nova-CAN will be based primarily on:

- **One-to-one messages** (one-way)
- **Services** (two-way)

Additionally, one-to-many communication is available in specific scenarios (e.g., system-wide halt messages and telemetry from devices). This differs from the OpenCyphal protocol where only services contain both destination and source node IDs. The modification was made due to the resource-constrained microcontrollers used in NovaRover devices, which require hardware-level CAN filtering to reduce load.

2.1 Definitions

Node: Each device is a node with a unique 6-bit device ID. Valid values are $[1 - 127]$.

Message: A one-way, one-to-one or one-to-many communication.

Service: A two-way, one-to-one request/response communication.

Subject: Messages and services are sent/received/invoked on subjects. A subject is a 9-bit identifier, allowing:

- 512 send subjects
- 512 receive subjects

Each subject is of a predefined data type (see Section 4). Protocol Subjects (defined in Section 5) must be implemented by all nodes. Each node also has:

- Subscribed subjects
- Published subjects
- Parameter subjects

These are defined by device interfaces (see *Device Interface* section).

3 CAN-ID Layout

The bit layout for a CAN-ID is shown in Figure 1.

Value	Priority [0,7]			Service Flag	Request Flag	R	Subject ID									Destination ID							Source ID						
CAN ID bit	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 1: CAN-ID Bit Layout

Field	**Width (bits)**	**Valid Values**	**Description**
Priority	3	$[0-7]$	Message priority from 0–7 (see *Priority*).
Service Flag	1	$[0, 1]$	0 for message, 1 for service.
Request Flag	1	0 for messages $[0, 1]$ for services	If service flag is 0, request flag must also be 0 (frame discarded otherwise). If service flag is 1, request flag $= 1$ for service request, 0 for service response.
R (Reserved)	1	0	Reserved for future use.
Subject ID	9	$[0-511]$	Represents a specific data type (see *DSDL section*). Per-node, allowing up to 512 subjects (messages/services).
Destination ID	6	0 reserved for multicast. $[1-127]$ for node-specific messages/services	Destination node ID. $0 =$ multicast message (multicast services are invalid). Frames with destination ID 0 and service flag 1 are invalid.
Source ID	6	$[1-127]$	Source node ID (0 is invalid).

4 Message Data Layout and Encoding

4.1 Frame Header

All frames shall have a one-byte header as seen in Table 2

Table 2: Header Byte Structure

Bit	Field	Single-frame transfers	Multi-frame transfers
7	**Start of transfer**	Always 1	First frame: 1, otherwise 0.
6	**End of transfer**	Always 1	Last frame: 1, otherwise 0.
5	**Toggle bit**	Always 1	First frame: 1, then alternates;
4	**Transfer-ID**	Modulo 32 (range [0, 31])	
3			
2			
1			
0			

Start of transfer : Flag for the start of a transfer. This allows detecting the start of a multi-frame message.

End of transfer : Flag for the end of a transfer. This allows detecting the end of a multi-frame message.

Toggle bit : Toggles on alternate frames of multi-bit messages. Used for deduplication of messages.

Transfer-ID: Cyclic modulo 32 transfer ID for the subject. Increments for every transfer. This is used for multiframe reconstruction and service call matching to allow multiple service calls from the same client to server. Transfer ID for a service response is not incremented, it is copied.

4.2 CRC

Multi-frame transfers require a CRC to check data-integrity across the entire message. This shall be added to the final frame of the message. *(TODO: define the exact CRC used)*

4.3 Data Definition and Format

All messages shall be defined through a message description language. Version 0 of this specification will rely upon the *OpenCyphal Data Structure Description Language*, described here in it's specification, and the open-source nunavut transpiler to generate C, C++ and Python3 types.

5 Protocol Subjects

(To be completed)

6 Device Interface

(To be completed)

7 System Description

(To be completed)