VLCB Versatile Layout Control Bus

CAN Service Specification

Service# 3 CAN

Version 1.1, March 2024, for Service version 1

Compatible with CBUS ® 4.0 Rev 8j

0.1 Table of Contents

0.2 Document History

Date	Changed by	Summary of changes	Service version
18th October 2022	Ian Hogg M.5144	Initial document	1
14 April 2023	Ian Hogg M.5144	Changed name to VLCB	1
10 February 2024	Ian Hogg M.5144	Updated section on Power on self tests and made tests optional	1

1. VLCB Services

This document describes the VLCB CAN service. This service is an optional addition to the VLCB Minimum Node Specification.

Modules wishing to use the VLCB CAN functionality shall conform to this specification.

2. CAN specific specification

CAN is a network system used in industrial and transport applications. It provides broadcast communications functionality and is particularly suited to a layout control bus due to its hardware support, and availability. VLCB is able to use CAN for multipoint to multipoint communications. See: https://en.wikipedia.org/wiki/CAN_bus

It sends data packets consisting of a header, some flags, a length-field, and an 8-byte data part. The header is involved in arbitration to prevent bus collisions. The flags include a RTR bit that requests replies from other nodes, and is used below.

Using the CAN bus imposes some specific requirements, such as packet-header uniqueness. The CAN Service guarantees this by a CAN-identifier, CANID, that is unique to each module, and so identifies each module uniquely on a bus segment.

2.1. CANID uniqueness

A node is required to have a unique CANID on each CAN bus segment (see Section 4.1 for the CAN data format). This number is part of the identifier field, which is used in the CAN arbitration scheme.

The CAN bus relies on hardware detection and arbitration of collisions between messages simultaneously transmitted by two or more nodes. This is done in hardware by actively comparing the header-part of the messages bit-by-bit, and the node which is transmitting the lower-priority message discontinuing its transmission.

To ensure a unique-header, VLCB assigns a unique CANID to each module and includes it in its message-headers. Since a module can only transmit a single message at a time, this guarantees that its headers are unique from all other messages on the bus, as per the CAN bus requirement.

Each module initially obtains its unique CANID by the process of *self-enumeration* (see Section 3.1) early in start-up, but it can also, in the future, retrieve this CANID from storage on start-up. As a consequence, a module is also required to detect this same CANID, used by another node, and to resolve this conflict by re-running the self-enumeration procedure.

2.2. Dependencies on other services

The CAN service depends upon the mandatory Minimum Node Service.

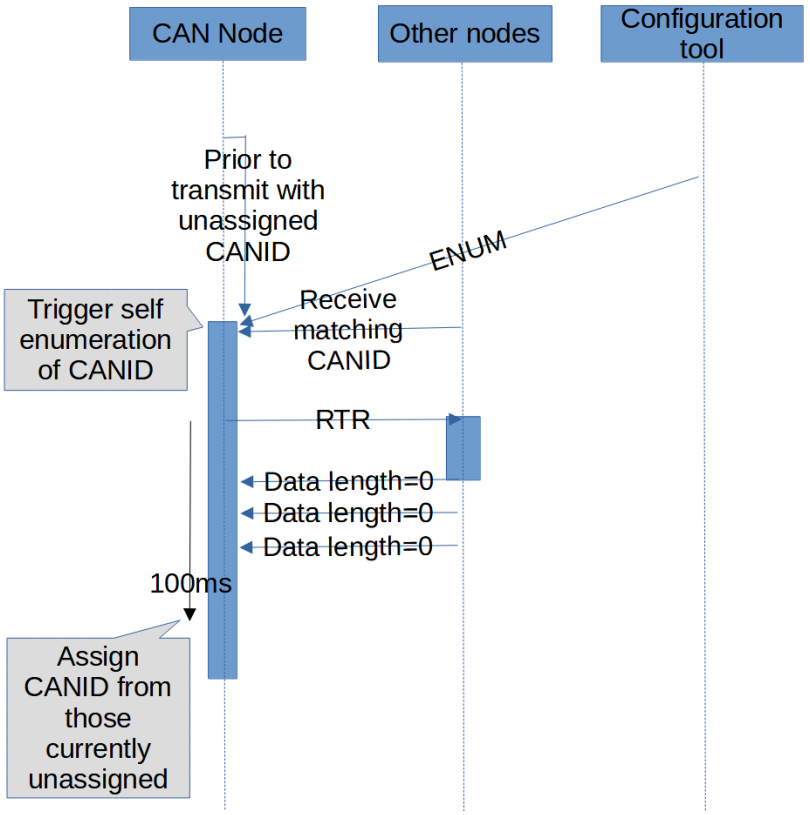
3. CANID processes

Valid CANIDs are within the range 1..127. A module may obtain an unused CANID using this process.

3.1. CANID self-enumeration

In order to obtain an unused CANID the module evokes all nodes to list their CANIDs, and chooses an unused one. To do this it must transmit a RTR frame without a data part, and wait for 100 ms, during which time it receives zero length data-frame replies from other modules. These zero-length data frames do contain the other modules' CANIDs in their header, and the module undergoing the enumeration process marks these CANIDs as already being taken. After 100 ms the module can then use one of the CANIDs which is not currently used by another module.

As a consequence, a module may receive a RTR frame at any time, and must respond with a zero length data frame containing its own CANID.



Zero is a reserved, invalid CANID and is not available to MMS compliant modules during CAN operation.

3.2. CANID initialisation

A fresh, uninitialised module will need to obtain a CANID using self-enumeration before it transmits any VLCB messages. The module must not perform self-enumeration immediately upon power up. It is recommended to perform self-enumeration using one of the following strategies:

1. Before first message transmission, typically RQNN.
2. When the push button is pressed, typically when entering SETUP mode.
3. At a random delay after power up, care must be taken to avoid multiple modules doing self-enumeration at the same time.

Once a valid CANID has been obtained it should be stored in non-volatile memory so that it is reused on subsequent power up of the module.

3.3. Auto conflict resolution

If a node receives a CAN frame which uses the same CANID as itself then the node should perform auto-conflict resolution by performing self-enumeration to obtain a new, unique CANID.

3.4. Diagnostics

Certain operations and situations involving the CANID involve diagnostics, see the Diagnostics section.

4. Mapping of VLCB to CAN

4.1. CAN Frame format

CAN Frame Format for VLCB Messages																															
CAN Header																															
CAN ID (11-bits)											DLC	EXT	RTR		DATA (bytes)															
10	9	8	7	6	5	4	3	2	1	0	4-bit	1-bit	1-bit		0	1	2	3	4	5	6	7									
Major Priority		Minor Priority		CANID							DLC	EXT	RTR		Opcode	Variable use, dependent on the Opcode.															
00=High … 11=Low		00=High … 11=Low		1-127*							0-8	0-1	0-1																		
Example: Accessory ON (ACON) Format, normal priorities																															
2		2		0x34							5	0	0		0x90	NN		EN		-		-		-						

DLC = number of data bytes in frame: 0-8.
EXT = Extended Header Flag: 0=Standard (11-bits), 1=Extended (29-bits).
Note: VLCB messages use Standard headers.
RTR = Remote Transmission Request; used in CANID Enumeration.
* - Note that CANID above 99 are often permanently assigned.

4.2. Bit timing

The VLCB CAN bus operates at a bit rate of 125 kbits/sec.

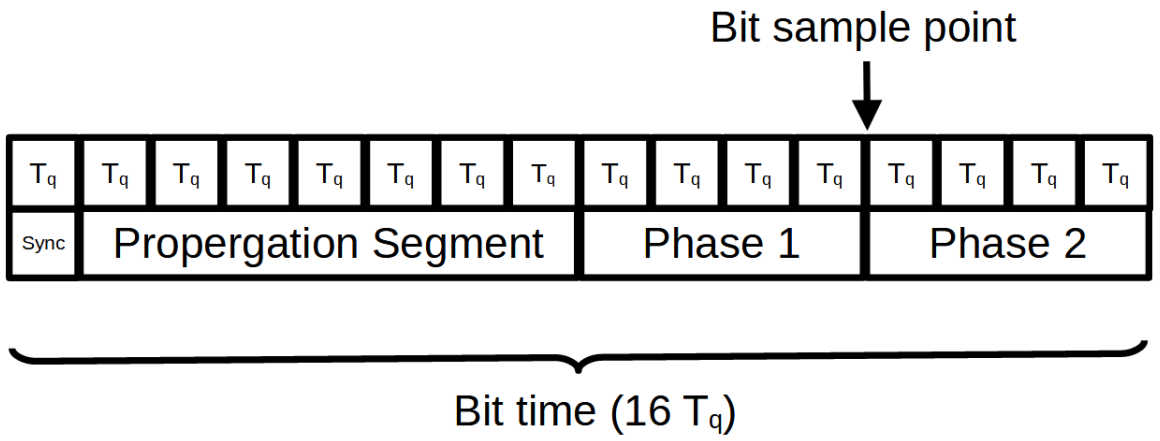
Different controllers will have differing setup values to determine the sampling point within a CAN bit. To do this, the CAN-bit is broken into a number of time-quanta (T_q), and into the following segments, in order: sync-segment, propagation-segment, Phase1-segment, and Phase2-segment.

The length of these segments, in terms of T_q , are programmed into the controllers registers. The following parameters work well for the common PIC microprocessors:

- Based on the base clock freq and divider, T_q = 500ns, so
- 1 CAN bit-time equals $16T_q$
- Sync-segment = 1 T_q
- Propagation time = 7 T_q (this is based on bus length);

- Phase1 = 4 T_q
- Phase2 = 4 T_q

This results in giving a total bit time of (1+7+4+4) 16 T_q.



This supports a bus length of 310 meters. While the values above are programmed into the controller registers, often a value one less will be the actual value programmed. Also note that some controllers combine the sync-segment into a Tseg1, so read the manual!

5. Module Self-Test capabilities

The CAN Service specification does not mandate and automatic power on tests nor any user requested tests however it is realised that some hardware specific tests may be beneficial and therefore it is left up to the module designer or the developer of software libraries to implement any test facilities that the user may find useful to help diagnose any build problem or operational problems.

6. CAN Priority

The top 4 bits of the CAN identifier are used for message priority purposes.

VLCB does not require that modules ramp up the CAN frame priority if a frame fails to be sent.

The VLCB Opcode Specification has defined a specific message priority for each opcode. The message priority (LOW, NORMAL, ABOVE, HIGH) are assigned so that command > response > event.

The message opcode priority is mapped onto the the can priority as follows:

CAN Priority value	Priority	Usage
0000	Highest	Self Enumeration process
0001		Reserved
0010		Reserved
0011		Reserved
0100		HIGH message priority
0101		ABOVE message priority
0110		NORMAL message priority
0111		LOW message priority
1000		Reserved
1001		Reserved
1010		Reserved
1011		Reserved
1100		Reserved
1101		Reserved
1110		Reserved
1111	Lowest	Reserved

7. Summary of Opcodes

The following opcodes have traditionally been used with the CAN interface however since VLCB modules are required to support CANID self enumeration and CANID auto conflict resolution then these opcodes are no longer required.

Request to Module			Module's Response			Use/meaning
Name	**OPC**	**Parameters**	**Name**	**OPC**	**Parameters**	
CANID	0x75	NN, CAN_ID	**WRACK**	0x59		Set a CANID
			CMDERR	0x6F	Error number	Error response
			GRSP	0x8F	CANID, Error	Error response
ENUM	0x5D	NN	**NNACK**	0x52	Current NN	Successful CANID self-enumeration has been completed
			CMDERR	0x6F	Error number	Error response
			GRSP	0x8F	ENUM, Error	Error response

VLCB modules should ignore these opcodes.

8. Service Specific Modes

No additional operating modes are specified by the CAN service.

9. Service Specific Status Codes

No additional GRSP status codes are specified by the CAN service.

10. Service Specific Diagnostic Data

10.1. DiagnosticCodes

The following DiagnosticCodes for the CAN service are supported:

0x00: return a series of DGN messages for each of the CAN services' supported data.

0x01: return CAN RX error counter

0x02: return CAN TX error counter

0x03: return last CAN status byte (TBA)

0x04: return Tx buffer usage count

0x05: return Tx buffer overrun count

0x06: return TX message count

0x07: return RX buffer usage count

0x08: return RX buffer overrun count

0x09: return RX message counter

0x0A: return CAN error frames detected

0x0B: return CAN error frames generated (both active and passive ?)

0x0C: return number of times CAN arbitration was lost

0x0D: return number of CANID enumerations

0x0E: return number of CANID conflicts detected

0x0F: Number of CANID changes

0x10: Number of CANID enumeration failures

Please refer to the specification of RDGN and DGN opcodes.

10.2. Diagnostic Payload Data Return

The following additional RDGN diagnostic data numbers are specified by the CAN service:

Diagnostic Code	Name	Diagnostic Byte1	Diagnostic Byte2	Explanation
0x01	RXERRCNT	Counter Hi	Counter Lo	This word is a 16 bit count of the number of CAN receive errors that occurred since power up. Note the module designer may have to implement these counts in software if not maintained by the CAN hardware. It is up to the implementer to designate what is a CAN RX fault.

0x02	TXERRCNT	Counter Hi	Counter Lo	This word is a 16 bit count of the number of CAN transmit errors that occurred since power up. Note the module designer may have to implement these counts in software if not maintained by the CAN hardware. It is up to the implementer to designate what is a CAN RX fault.
0x03	ERRSTAT	Bitfield	reserved	Bits indicating error states.
	Bit 7: ARBF: after 10 arbitrations failure this bit is set Bit 6: ALIGNF: a CAN alignment error occurred Bit 5: BITSTUFF: a CAN but stuffing failure occurred Bit 4: TXUF: A unspecified failure to transmit occurred Bit 3: RXUF: an unspecified receive failure occurred Bit 2: BAUDF: a baud rate mismatch or parity or other speed related error occurred Bit 1: EXXACT: The mode has switched to error active (lo indicated error passive) Bit 0: BUSOFF: the node has experienced a bus off state since power up (i.e. maintained across bus-off states).			
0x04	TXBUFUSE	Counter Hi	Counter Lo	Current number of transmit buffers in use.
0x05	TXBUFOVER	Counter Hi	Counter Lo	Count of the number of transmit buffer overruns.
0x06	TXMESS	Counter Hi	Counter Lo	Count of the number of messages transmitted.
0x07	RXBUFUSE	Counter Hi	Counter Lo	Current number of receive buffers in use.
0x08	RXBUFOVER	Counter Hi	Counter Lo	Count of the number of receive buffer overruns.

0x09	RXMESS	Counter Hi	Counter Lo	Count of the number of messages received.
0x0A	RXERR	Counter Hi	Counter Lo	Count of CAN error frames detected. If a node can detect both error active and error active frames, this is a running 16 bit counter of the total number of such frames received since power up.
0x0B	TXERR	Counter Hi	Counter Lo	Count of CAN error frames transmitted. Count of both error active and error passive frames generated by this node since power up.
0x0C	ARBLOST	Counter Hi	Counter Lo	Count of CAN the number of times CAN arbitration was lost.
0x0D	IDENUM	Counter Hi	Counter Lo	Count of the number of CANID enumerations
0x0E	IDCONF	Counter Hi	Counter Lo	Count of the number of CANID conflicts detected
0x0F	IDCHANGE	Counter Hi	Counter Lo	Count of the number of CANID changes.
0x10	IDFAIL	Counter Hi	Counter Lo	Count of the number of CANID enumeration failures

11. Service Specific Automatic Power-up Tests

It is recommended that upon power up the module shall attempt to confirm that its own hardware is functioning correctly and is correctly connected to the CAN bus. Some potential tests that could be performed are:

CAN internal loopback	Some CAN hardware may not allow this.
CAN external loopback	This may require dedicated hardware to connect the TX data to RX data.

CAN transmit acknowledge test	Some CAN hardware may not provide an indication of transmit success.

If any test indicates a failure or problem then this must be indicated to the user. The way in which the test results are reported is module dependent.

Given that these tests are very dependent upon the hardware capability all tests are optional.

12. Documentation

Any test capabilities supported by the module shall be documented.

13. Service Data

13.1. Parameters

The following parameters are associated with the CAN service:

Address	Param#	Name	VLCB should set these values
0x829	10	Protocol	Set to 1 for CAN

13.2. ESD data bytes

The CAN service does not currently pass any information back in the ESD data bytes.