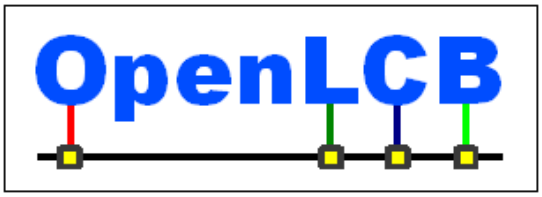


OpenLCB Technical Note	
Memory Configuration	
Feb 17, 2015	Adopted

1 Introduction

This Technical Note provides background for the association Memory Configuration Standard.

The protocol is called “Memory Configuration” because it makes the configuration information in the node look like it's stored in linear memory spaces.

- Don't assume that the information actually has to be stored the same way, it is accessed through this protocol, as linear memory. The node can remap the information that's being read or written into whatever internal organization it needs.
- Don't assume that no other configuration protocol exists. Some day, for example, there might be a “file access configuration protocol” where a node pulls predefined configuration information from some central store of “files”. Or something else. There's already the teach/learn method of configuring events.
- Don't assume that this is only used for persistent configuration information. The protocol is already being used for e.g. retrieving the CDI, and for providing a simple debug capability that allows programmers the ability to read (and even write) raw node RAM. It can be used for other things in the future using the memory space mechanism.

2 Annotations to the Standard

2.1 Introduction

Note that this section of the Standard is informative, not normative.

2.2 Intended Use

Note that this section of the Standard is informative, not normative.

2.3 Reference and Context

For more information on format and presentation, see:

- OpenLCB Common Information Technical Note

2.4 Message Formats

This section is about helping implementors see how to decode the message format. It's not normative.

Configuration messages use a specific datagram format consisting of the datagram type byte, followed by a single byte combining the operation “Command Type” field and flags. This is then followed by data in an operation-specific format.

Byte 0	Byte 1	Byte 2-5	Byte 6 (Optional)	Remaining Bytes
Datagram Type	Command Type	Starting Address	Address Space	Command Specific

Byte 1 has a number of decodable fields:

Byte 1 (Command Type)	
Bits 7..6 – Command Type	0b00 – Write operations 0b01 – Read operations 0b10 – Control operations 0b11 – Not used / reserved
Bit 5 – Stream/Datagram	0b0 – Use Datagrams 0b1 – Use Streams
Bit 4 – Command/Reply	0b0 – Command 0b1 – Reply
Bit 3 – (Command) Under Mask **Bit 3 – (Reply) Fail/OK**	0b0 – Not under mask 0b1 – Under mask 0b0 – Okay 0b1 - Fail
Bit 2 – Reserved	0b0 – send as zero, check on receipt
Bits 1..0 – Address Space	0b00 – Address space in byte 6 0b01 – Address space 0xFD 0b10 – Address space 0xFE 0b10 – Address space 0xFF

2.4.1 Address Space Size

The four-byte address allows directly addressing 4GB of data. The use of address spaces (see 2.4.2 Address Space Selection) allows direct access to 1TB of data.

The large address range removes the need for address registers and other non-idempotent accesses when accessing e.g. sound information in a large memory.

2.4.2 Address Space Selection

Although a 32-bit address space is large enough to cover combined uses of memory, it can be more convenient to consider separate address spaces in the node. This can also be considered to be a top digit in a global address space, if desired, but note that the separate address spaces may cover the same memory objects, e.g. “all memory” and “Event ID configuration” spaces may reference the same physical (or virtual) location in memory.

2.4.3 Generic Error Handling

An unknown command in byte 1 may result from a command that is not defined by the standard, but it may also result from a command the that standard defines, which the node being accessed does not support. The error code to send is not specified, however, the recommended error code to be sent in the OpenLCB Datagram Transport Datagram Rejected message is any one of the following:

Permanent Errors from the OpenLCB Message Network Standard:

- 0x1000 – Permanent error, not further specified
- 0x1040 – Not implemented, not further specified.
- 0x1050 – Not implemented, subcommand is unknown
- 0x1060 – Not implemented, Datagram-type, Stream-type, or command is unknwon
- 0x1070 – Not implemented, unknown MTI, or Transport protocol (datagrams/streams) is not supported

It is possible that OpenLCB Datagram Transport is not supported by the node, and by extension OpenLCB Memory Configuration. If this is the case, the likely result is an OpenLCB Optional Interaction Rejected message as defined in the OpenLCB Message Network Standard with one of the permanent error codes listed above.

Additionally, the Datagram Rejected message may be received due to an OpenLCB Datagram Transport layer error. This does not mean the Memory Configuration generated the error, and could be due to a temporary inability to allocate a necessary buffer at the Datagram Transport layer, etc...

2.4.4 Read Command

This is a request to read an address space. If the address space does not exist, a Datagram Rejected message may be sent with a permanent error. The likely error code would be:

0x1080 – Invalid arguments. Some of the values sent in the message fall outside of the expected range, or do not match the expectations of the receiving node.

The address space may be valid, however, the node may be busy or locked. In this case, the Datagram Rejected message may be used, likely with a temporary error code (see OpenLCB Message Network Standard). The requesting node may, or may not, try again later.

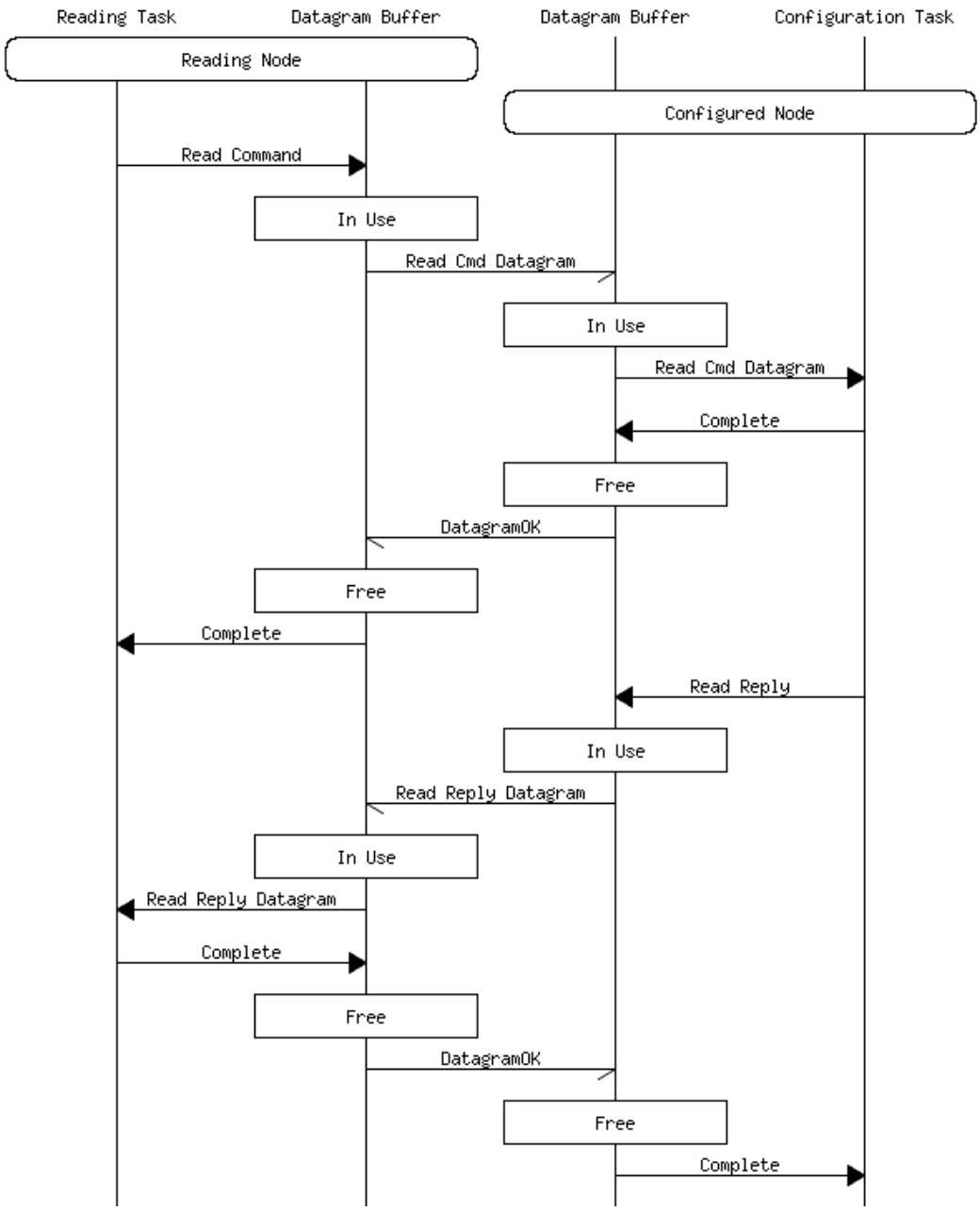
If the memory configuration protocol is being used to control e.g. DCC CV reads and writes, those operations can take a very long time, 30 seconds or more. The Datagram OK message optionally specify a timeout interval in order to provide guidance as to how long the read may take before considering the read to be timed out. The timeout interval may in turn, but doesn't have to be, used to indicate in-progress status.

2.4.5 Read Reply

This message is sent in response to a Read Command which was previously responded to with a corresponding Datagram OK message having had the Reply Pending bit set. If the Read Command was previosuly responded to with a corresponding Datagram Rejected message, no Read Reply message is required or expected.

A Read Reply is not considered to be in error unless it responds with zero bytes of data. In this case, a failure should be indicated with an appropriate error code as defined by the OpenLCB Memory Configuration Standard and/or the OpenLCB Message Network Standard.

The figure shows a typical read operation. The Read Command is carried to the device being configured by a datagram, the read operation takes place, and the results are returned in another datagram.



2.4.6 Read Stream Command

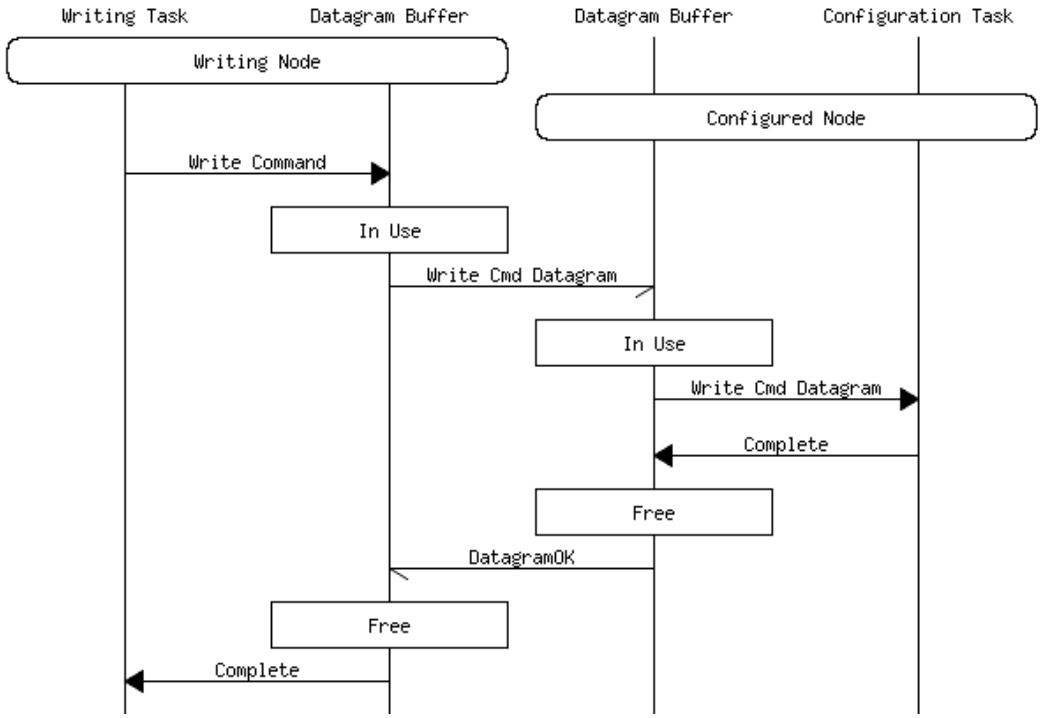
Stream support within the OpenLCB Memory Configuration Protocol is considered experimental. A version of this Technical Note and corresponding Standard containing updated information on the usage of streams will likely be adopted at a future time.

2.4.7 Read Stream Reply

Stream support within the OpenLCB Memory Configuration Protocol is considered experimental. A version of this Technical Note and corresponding Standard containing updated information on the usage of streams will likely be adopted at a future time.

2.4.8 Write Command

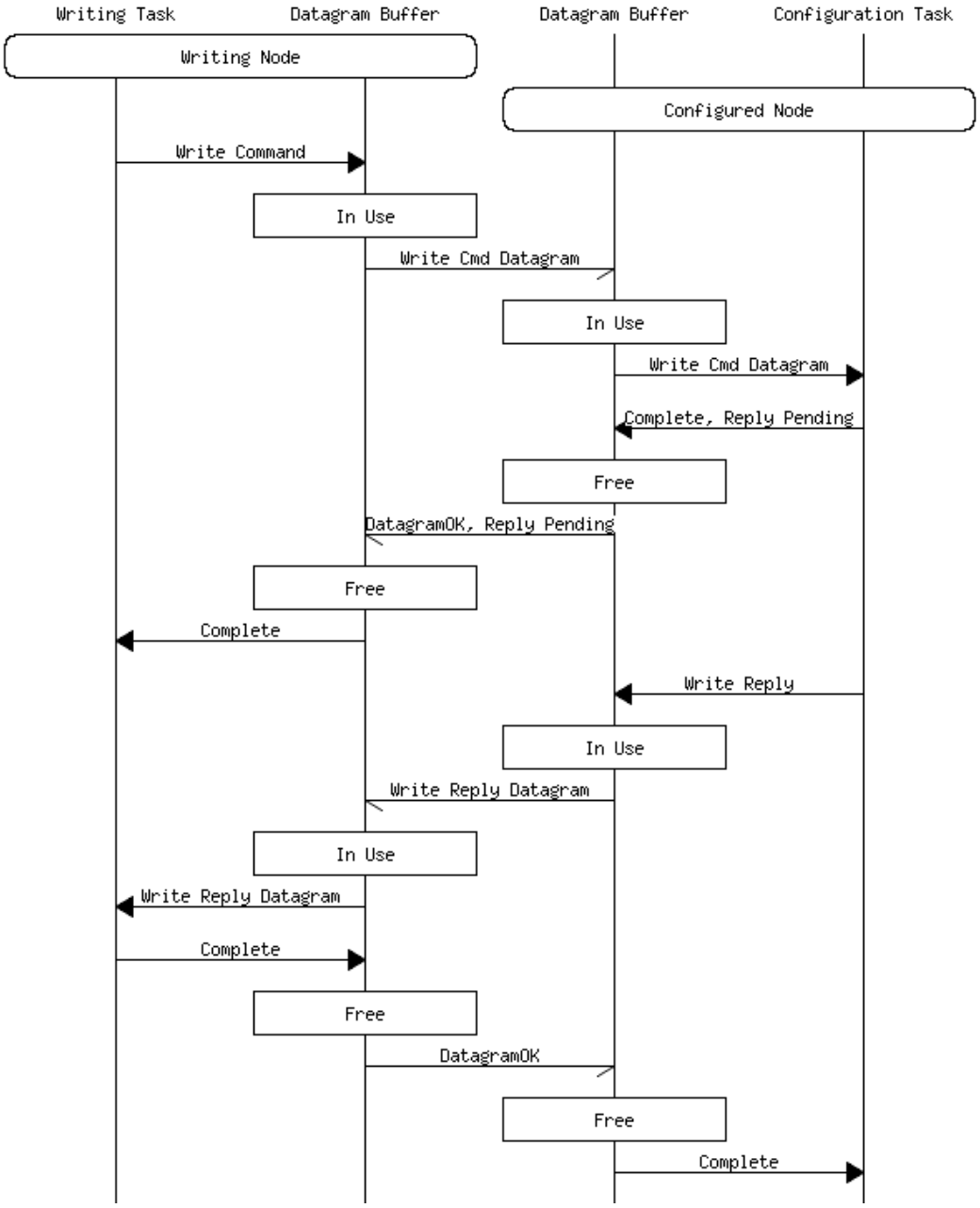
Most writes are assumed to always complete immediately and without error. Writes into memory are certainly like that, but writes to DCC CVs might not be. Into-memory writes are well served by "when the positive reply to the datagram arrives, the write completed successfully". That requires no extra traffic to handle other cases.



A typical memory write operation. The Write Command and data are carried to the node being configured in a single datagram. Since the write operation succeeds immediately, the only reply needed is the datagram reply.

2.4.9 Write Reply

For the less-common case of writes that might fail and/or take a long time, the Datagram OK reply to the Write datagram can have the Reply Pending bit set\, followed by a Write Reply when the operation is done. That Write Reply, in turn, informs the requester when the operation was complete, and whether it completed OK.



A delayed response, either because the write takes significant time or could fail for some reason. The datagram reply to the Write Command carries the reply pending bit set, and a reply datagram is send from the configured node back to the configuring node with the reply.

2.4.10 Write Under Mask Command

A write under mask differs from a standard write only in so far as it uses a stream of alternating bytes, the first of which is a mask and the second of which is a value. Otherwise a Write Under Mask Command sequence is identical to a standard Write Command sequence.

The primary use for this is to set/clear individual bits within a larger addressable type such as a byte, or multi-byte word.

2.4.11 Write Stream Command

Stream support within the OpenLCB Memory Configuration Protocol is considered experimental. A version of this Technical Note and corresponding Standard containing updated information on the usage of streams will likely be adopted at a future time.

2.4.12 Write Stream Reply

Stream support within the OpenLCB Memory Configuration Protocol is considered experimental. A version of this Technical Note and corresponding Standard containing updated information on the usage of streams will likely be adopted at a future time.

3 General

3.1 Environment of Protocol

3.1.1 Requirements

- Nodes must carry enough context that a stand-alone configuration tool can provide a useful human interface without getting any data from an external source, e.g. needing an Internet download to handle a new node type.
- It must be possible to configure a node entirely over the OpenLCB, without physical interactions, e.g. pushing buttons. This configuration must be compatible with local configuration, including e.g. the Blue/Gold method.
- It must be possible to configure one or more nodes while the rest of the OpenLCB is operating normally.
- It must be possible to read, store, and reload the configuration of a node.

3.1.2 Preferences

- Small nodes shouldn't need a lot of processing power, e.g. to compress or decompress data in real time. Memory usage should also be limited, but is a second priority.
- Configuration operations should be state-less and idempotent to simplify software at both ends.
- Multiple independent configuration operations can proceed at the same time. Multiple devices should be able to configure separate nodes at the same time. Multiple overlapping reads of the same node should be possible. There should be a method to coordinate separate configuration operations to simplify configuration software.
- For efficiency, atomic reads or writes of small amounts of data should fit into a single-frame CAN datagram. Single-bit writes also add efficiency.
- Multiple address spaces make it easier to handle multiple types of data.
- For large transfers, it's desirable to be able to use streams. Not all nodes support them, though, so it must be possible to Inquire about capabilities.
- Addresses should be four bytes. Two address bytes is a failure of imagination.

3.1.3 Design Points

Basic configuration is done with datagrams, which can carry 64 bytes of data to read or write.

For efficiency, writes don't need any reply except the datagram response. By sending that only after the write is complete, including non-volatile memory delays, the written node can control the transfer rate.

Read operations must return data in a separate datagram. If this carries the address, etc, in addition to the data, the operation is idempotent.

Read and write operations can address separate kinds of information in the node via specifying specific address spaces. Three of these have specified uses, and the reset are optional tools for future expansion.

There are a large number of possible configuration options, and more may be developed in the future: Reset, identify, etc. Not all nodes will implement all methods, so a query operation is needed so that tools can know what they can do.

Can't require that nodes do anything in particular to put changes into effect. Some will do it right away, some will require a reset, etc. Need flexibility for node developer, yet consistency for configuration tool builder.

3.1.3.1 Large read via stream

Stream support within the OpenLCB Memory Configuration Protocol is considered experimental. A version of this Technical Note and corresponding Standard containing updated information on the usage of streams will likely be adopted at a future time.

3.1.3.2 Large write via stream

Stream support within the OpenLCB Memory Configuration Protocol is considered experimental. A version of this Technical Note and corresponding Standard containing updated information on the usage of streams will likely be adopted at a future time.

3.1.3.3 Performance Note

Small nodes may have only one datagram receive buffer and one datagram transmit buffer. In that case, sending multiple reads can result in poor performance if the replies go after the next request, e.g.:

Read [d] →

← datagram reply

← Read Reply [d]

Read [d] →

datagram reply →

← datagram reply

- ← Read Reply [d]

datagram reply →

Note that the 2nd read command was sent before the reply to the Read-Reply datagram was sent. This is valid, but nodes with only one buffer may not be able to handle that Read until the transmit buffer has been emptied by an acknowledgement. In that case, the sequence may become e.g.:

Read [d] →

← datagram reply

← Read Reply [d]

Read [d] →

← datagram negative reply (reject, no buffer, please resend)

datagram reply →

Read [d] (retransmit) →

← datagram reply

← Read Reply [d]

datagram reply →

which is significantly less efficient. Better to send the positive acknowledgement to the read reply datagram before sending the next read command.

3.1.4 Delays Due to Non-Volatile Memory

Some microcontrollers can't continue to operate while writing configuration information to non-volatile memory.

If the CAN buffering is sufficient (at about 200 usec per buffer) for the node to become active at the end of the memory operation and process buffered frames at the full rate, there's no issue.

If a node has missed one or more frames, it's possible that some state interaction has started and the node is inconsistent. For example, a RIM/CIM sequence could have started or even finished.

If the node misses one or more frames, the CAN controller needs to flag that and bring it to the attention of the node's microcontroller. The node then needs to emit an “Initialization Complete” message so that other nodes realize that this node may not have complete state information.

The node should also defer the reply to a write datagram until after the write is complete, to make it less likely the next-in-sequence operation arrives during dead time. Nodes doing the configuring should limit the amount of traffic they send to the node-under-configuration to reduce the need for e.g. datagram retransmission.

There are bits that allow the node to specify what size writes are allowed. These can be used to force the configuring node to do operations in the most efficient way, e.g. to minimize dead time during writing.

The transfer buffer size for stream access is negotiated. By requiring a transfer size equal to or smaller than the memory write size (page size), the node can ensure that the stream will pause during a write operation.

3.1.5 Large Volume Operations

The stream protocol is meant for large reads and writes, but the datagram protocol can also work well on a single CAN segment. The difference in performance comes from the (potential) larger datagram buffer size.

All nodes support datagrams for configuration; not all support streams. So a least-common-denominator configuration tool would use sequences of datagrams for even large transfers. Because the need for reply, short datagrams are not particularly efficient. In the limiting case, you can only write two bytes per frame exchange. The sending node should look at the Get Configuration reply and use the largest available size.

On the other hand, if non-volatile memory timing requires that write operations to a node use a 64-byte or smaller stream buffer size, then datagrams are a more efficient method than streams. In that case, the node should indicate that stream-write operations are not supported in its reply to Get Configuration. Since read operations don't have the same timing issues, streams may still be useful in that case.

Table of Contents

235

