



OpenLCB Technical Note	
OpenLCB Message Network	
Mar 19, 2012	Preliminary

## 1 Introduction

This explanatory note contains informative discussion and background for the corresponding “OpenLCB CAN Message Network Specification”. This explanation is not normative in any way.

## 5 2 Annotations to the Specification

This section provides background information on corresponding sections of the Specification document. It's expected that two documents will be read together.

### 2.1 Introduction

### 2.2 Intended Use

### 10 2.3 References and Context

### 2.4 States

The Uninitialized state is only occupied when the node is first starting up. At present, there's no way to deliberately return to it.

### 2.5 Message Format

### 15 2.6 Messages

(Note that Node ID in the data part of several messages is sent in full 48-bit format in all wire protocols, specifically including CAN, even if an alias or alternate form is available elsewhere in the message)

### 2.7 Interactions

20 All nodes must be able to take part in all standard interactions.

#### 2.7.1 Node Initialization

Newly functional nodes, once their start-up is complete and they are fully operational, shall send an "Initialization Complete" message and enter Initialized state.

- There is no guarantee that any other node is listening for this. No reply is possible.
- Nodes must not emit any other OpenLCB message before the “Initialization Complete” message.

Sending the IC message is required to insure that higher-level tools are notified that they may start to work with the node.

### 2.7.2 Duplicate Node ID Discovery

- 30 OpenLCB nodes must have unique node IDs. The Frame Transfer protocol will detect duplicate node IDs on a single OpenLCB segment, e.g. a CAN bus, but is not intended to detect duplicate node IDs across multiple segments. To detect duplicates across the entire connected OpenLCB, all OpenLCB nodes must indicate an error if they detect an incoming message with a Source Node ID equal to their own. If possible, they should indicate it at the board itself using a light or similar. If possible, they should emit a PCER message with the “Duplicate Source ID detected” global event, which will carry the duplicate node ID in the Source Node ID field.

40 After sending the “Duplicate Source ID detected” global event, the node should not transmit any further “Duplicate Source ID detected” messages until reset because this message will be received at the other duplicate-ID node(s), resulting in additional “Duplicate Source ID detected” global events and causing a possible message loop. (Optionally, could allow to send again after e.g. 5 seconds)

To further improve the reliability of this detection, OpenLCB nodes should, but need not, emit a Verified Node ID message every 30 to 90 seconds. As an implementation detail, it's recommended that CAN-attached nodes use their NIDa to pick that interval so that messages don't bunch up.

### 2.7.3 Node ID Detection

- 45 Upon receipt of a Verify Node ID Number message addressed to it, or an unaddressed Verify Node ID Number message, a node will reply with an unaddressed Verified Node ID Number.

This can be used as check that a specific node is still reachable. When wire protocols compress the originating and/or destination NID, this can be used to obtain the full NID.

- 50 The standard Verify Node ID Number interaction can be used to get the full 48-bit NID from a node for translation. At power up each node must obtain a alias that is locally unique. Gateways will also have to obtain unique aliases for remote nodes they are proxying on to the segment.

### 2.7.4 Error Handling

#### 2.7.4.1 Reject Addressed Optional Interaction

- Node A receives an addressed message from Node B that carries Node A's NID.
- 55 • The MTI indicates the start of an optional interaction.
- If Node A does not want to take part in the optional interaction, it may send an Optional Interaction Rejected message addressed to Node B with the original MTI in the message content. There is no requirement that OIR be sent; the node may silently ignore the incoming message.

- 60 (The message content also contains an optional reason code and an optional data value. (Define use))

(This is written that sending the OIR in return is optional. This greatly increases the complexity of error handling on the originating node, though, as it can't assume that lost messages are a transient

error. It would be better to have this be a mandatory response to simplify that. Does it add much complexity cost to the nodes?)

65 (The phrasing is also in terms of an "optional interaction", which doesn't cover the case of "undefined interaction", e.g. an MTI that's not allocated. Since any given node doesn't actually know whether an MTI has been allocated or not (it might have been built a while ago, with new optional protocols added since then), this should be rephrased. I'll add a to-do item for that too.)

#### **2.7.4.2 Reject Unaddressed Optional Interaction**

- 70
- Node A receives an unaddressed message from Node B.
  - The MTI indicates the start of an optional interaction.

If Node A does not want to take part in the optional interaction, it silently drops the message without reply.

#### **2.7.4.3 Reject Addressed Standard Interaction Due to Error**

- 75
- Node A is taking part in an addressed interaction with Node B. Either node may be able to send the next message.
  - Some error condition prevents Node A from continuing the interaction.
  - To terminate the interaction, Node A sends a Terminate Due to Error message to Node B. It then resets it's state so as to no longer be taking part in the addressed interaction.

80 The message content contains the most recent MTI received in this interaction, a mandatory reason code and an optional data value.

Note that the specification doesn't say whether Node A or Node B started the interaction. It could have been either. Node A is just the name for the node that can't continue and wants to stop the interaction.

85 This is a very coarse mechanism. The "most recent MTI received" values is not always sufficient to determine which interaction is being referred to. Higher level protocols should define more focused and reliable mechanisms if they are likely to encounter errors.

### **3 MTI allocation method**

(This is just informative, not normative; it's the actual MTI values that are normative, not how they were picked)

90 (This could be here, or in a separate "MtiAllocations" note; there's already a draft of the separate note, which should be either made the sole repository of this info or moved here)

The specific MTI values are being documented in each protocol definition, and also here as a consistency check.

95 Because the MTI values are specified for each kind of message, the Standard just documents those results. This section of the Technical Note addresses the method for choosing specific values.

Some MTIs have additional status bits defined as part of the 2nd field. For example, there are two status bits associated with “Consumer Identified” which must be kept in the header since there is no room in the CAN data field. To simplify translation between formats, these are the low bits of the first byte after the MTI in a standard-form message.

100 Designers may wish to use CAN hardware filtering, but it can't be assumed to be present. We assign a single bit to indicate “simple node protocol” messages to make simple filtering possible.

The common Message Type Indicator (MTI) is a 16 bit quantity. Note that specific wire protocols may remap this.

- The most-significant 5 bits are reserved as 00110; nodes must send and check that value.

105 • The next 7 bits are used to indicate the message type.

- The top two of these are used to form static priority groups. A 0 bit is considered to have more priority (can be processed first), a 1 bit less priority (can be processed later). The MSB makes a larger statement about priority than the LSB of these two. Priority processing is permitted but not required. The priority group bits are part of the overall message type.

110

Note that the priority bit in the CAN frame is separate from the static priority field in the MTI format specification.

- The next bit is reserved as 0; nodes must send and check that value.
- The lower four bits indicate a specific type.

115 • The following bit is reserved as 0; nodes must send and check that value.

- The 2nd from-least-significant bit indicates that this message carries a destination node address (DID) when set to 1. Setting 0 means that the message is globally addressed. If a Destination Node ID (DID) is present, it is located at the start of the message content. The form of a Destination Node ID is defined by the particular wire protocol, but must be mappable to the full NID of the intended destination.

120

- The next-to-least-significant bit indicates this message carries a P/C Event ID field when set to 1. Setting 0 means that the message does not carry a P/C event ID. If a P/C Event ID is present, it's at the start of the message, except after the Destination Node ID, if present.

125 • The least-significant bit when set to 1 indicates this message carries a flag byte after the DID and/or EID determined by the above bits. The low bits of that byte can be relocated in CAN messages, see the definition of the CAN wire protocol.

We've chosen to allocate bit fields to make decoding simpler; if possible, aligned on nibble boundaries to make it easy to read as hexadecimal numbers. Note that, as a special dispensation for CAN, higher priority messages (MTIs with lower numerical values) may pass lower priority ones; this must be taken into account when designing protocols.

130

For OpenLCB messages, the variable field is used in two forms:

- Unaddressed messages – messages that don't have a destination address put the low 12 bits of the MTI in the variable field
- Addressed messages – messages that have a specific destination address put it in the variable field, and carry the MTI in the payload. This allows filtering.

135

The variable field is allocated:

Variable Field Bits 0-2	Variable Field Bits 3-14
Header Bits 2-4	Header Bits 5-16
OpenLCB Format	OpenLCB Variable Header Content
0x0700,0000	0x00FF,F000
0 0 0	MTI & additional information for “simple node” unaddressed messages
0 0 1	MTI & additional information for unaddressed messages other than “simple node” forms
0 1 0	(Reserved, must not be sent or accepted)
0 1 1	Long-form MTIs in data area
1 0 0	Destination Alias for datagram message non-last fragment
1 0 1	Destination Alias for datagram message last fragment
1 1 0	Destination Alias for non-datagram addressed messages
1 1 1	Destination Alias for Stream Data Send messages

140

Some MTIs have additional status bits defined as part of the 2nd field. For example, there are two status bits associated with “Consumer Identified” which must be kept in the header since there is no room in the CAN data field. To simplify translation between formats, these are the low bits of the first byte after the MTI in a standard-form message.

## 4 To Be Done

145

Needs a discussion of handling failed communications. The protocols are all designed to have error-responses (error reply to datagram; Option Interaction Rejected) instead of silently failing for directed

150

messages. Timeout logic is only necessary to handle transport failures and failures of nodes to execute the protocols, including e.g. when they power off in the middle of something. The Standard should say that “Nodes shall allow at least nnn msec for a reply to be received to their communications” and “Nodes shall reply to messages within mmm msec”. Then this TN can talk about timeouts, how you have to assume that you'll have to wait for a reply when arranging state machines and buffering, and the generic issues around recovering from an expected reply not being received. Any specific issues with timeout recovery can be discussed in the message-specific parts of the TN.

## Table of Contents

1 Introduction.....	1
2 Annotations to the Specification.....	1
2.1 Introduction.....	1
2.2 Intended Use.....	1
2.3 References and Context.....	1
2.4 States.....	1
2.5 Message Format.....	1
2.6 Messages.....	1
2.7 Interactions.....	1
2.7.1 Node Initialization.....	1
2.7.2 Duplicate Node ID Discovery.....	2
2.7.3 Node ID Detection.....	2
2.7.4 Error Handling.....	2
2.7.4.1 Reject Addressed Optional Interaction.....	2
2.7.4.2 Reject Unaddressed Optional Interaction.....	3
2.7.4.3 Reject Addressed Standard Interaction Due to Error.....	3
3 MTI allocation method.....	3
4 To Be Done.....	5