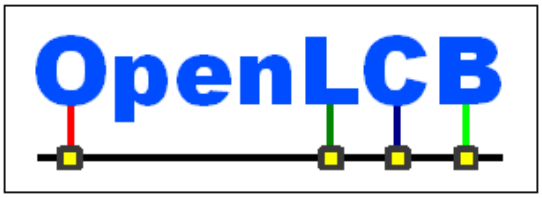

OpenLCB Technical Note	
Common Information	
May 14, 2012	Preliminary

1 Introduction

The OpenLCB Standards are independently normative. They are what they are, and they say what they say.

This Technical Note provides common background information that many be useful as you read existing Standards and Technical Notes, and may provide useful guidance as you write new ones.

This Technical Note is not normative in any way.

2 Data

Reserved quantities are created with a zero value unless otherwise specified. When processing a message, ignore reserved quantities unless otherwise specified. When transporting a message, transport reserved quantities unchanged. The zero value sometimes indicates a non-initialized value.

2.1 Numerical representation

Unless otherwise specified, OpenLCB fields are unsigned.

OpenLCB does not define a floating point representation (yet).

2.2 Byte sequences

Bytes are defined as 8 bits.

OpenLCB is, by default, big-endian. When sending multi-byte data, the byte containing the most significant bits is sent first. This is the same as the CAN header, Ethernet and the common internet protocols, but not the same as the Intel x86 architecture.

When a string of bytes is being documented or described, the first or most significant is labelled 0, the next is labelled 1, etc. This results in phrases like “byte 0” and “the first byte” referring to the same thing.

2.3 Bit sequences

The OpenLCB protocol descriptions use LSB 0 coding, where the the least significant bit in any word, byte or field is numbered 0, with bits to the left (toward the MSB) then given higher numbers.

Although “first bit” properly denotes the most-significant bit of the field being discussed, “second bit” refers to the bit adjacent to the first bit, etc, it is better to refer to “most significant bit” rather than “first bit”.

The CAN specification and some layout-level protocols, such as NMRA DCC and Digitrax Loconet, use LSB 0 coding. That's the primary motivation for the choice of LSB 0 for OpenLCB. Unfortunately, MSB 0, where the most-significant bit is labelled with 0, is what's used for many protocol specifications, including the RFC series of protocols and the Ethernet definition. It's unfortunate that there's no single convention that OpenLCB could adopt to be consistent with the entire world, but there isn't.

2.4 Strings

OpenLCB strings are sequences of plain ASCII 7-bit values. We want to retain the 0x80 bit in the first byte as a way of eventually indicating other codings. XML strings can start with a UTF BOM (either 0xEF, 0xFF or 0xFE in the 1st byte, since there's no need to support UTF-32BE or UTF-32LE), or the ASCII text for "<?xml" which starts with 0x3C.

OpenLCB does not prefer length-coded or null-terminated strings. Standards should specify which is used in each case.

OpenLCB uses newline, also known as line-feed (\n 0x0A) as the line-end character within strings. Carriage return (\r 0x0D) should be considered as general white space.

There are no specific tab settings. You cannot assume a tab is any particular number of spaces. You can assume it counts as non-null white space.

3 Presentation

OpenLCB documents use the prefix "0x" to indicate a hexadecimal value. Hexadecimal values are presented with capital letters: 0xAB not 0xab. A sequence of bytes only requires the "0x" prefix on the first byte: 0x12 34 56 78.

OpenLCB documents use the prefix "0b" to represent a binary value.

Octal representations are not used.

The boolean values are "true" and "false".

Constants should always include the full field length. A value for a 12-bit field should be written as 0x002 or 0b000000000010, not 0x2 or 0b10.

Commas can be used in numbers after decimal thousands (65,523), 16-bit double-bytes for hex constants (0x1234,4567), and four-bit nibbles in binary (0b1000,0000). Do not put a space after the comma.

Byte sequences, including both Unique ID (node ID) and Event ID values, should be shown in dotted-hex format e.g. "01.AB.34.01.CD.E3". It's recommended that leading zeros be provided on output, but not required on input. Dotted-decimal should not be used unless it's made very clear that that is the case.

3.1 Presentation of CAN Quantities

The 29-bit CAN extended header is presented as a single hex string: 0x0000,0000. The active bits are at the right side, so that the highest possible value is 0x1FFF,FFFF.

"sss" is used to represent the source node ID alias in a CAN frame.

"ddd" is used to represent the destination node ID alias in a CAN frame.

A sample CAN header with both source and destination node addresses might be: 0x1Fdd,dsss or 0x1Fdddsss.

The data content of a frame is presented as a sequence of individual bytes. A typical complete frame might be then 0x1FFFFsss 01 02 03 04.

3.2 Presentation of a OpenLCB message

OpenLCB messages are described in this format:

Message Type Indicator: The 16-bit value, typically in hex notation, of the full MTI

Destination address present: Whether this message includes a destination address for a specific node. If no, the message is global.

Simple subset: Whether the message is included in the "simple node subset" (defined where?)

Priority group: A number from 0 to 3 that describes the line priority of the message; 0 can be sent before 3.

Content: Description of data bytes, if any, included after fixed fields such as MTI and destination address.

CAN frame format: Typical CAN frame, including header and data bytes. Source node ID alias is represented by "sss". Destination node ID alias, if present, is represented by "ddd".

Note that this information is not completely independent. The MTI value depends on simple subset, priority subgroup, etc, values. We present the full set, at some risk of inconsistency, to make it easier to understand the containing Standard.

For more information, see the various "Message Networking" Standards and Technical Notes.

Example:

Message Type Indicator: 0x32F4

Destination address present: Yes

Simple subset: Yes

Priority group: 1

Content: Six bytes containing bits identifying the OpenLCB protocols supported by the sending node; see Section 6 below for coding.

CAN frame format: 0x18dd,dsss 32 E4 vv vv vv vv vv vv

Table of Contents

