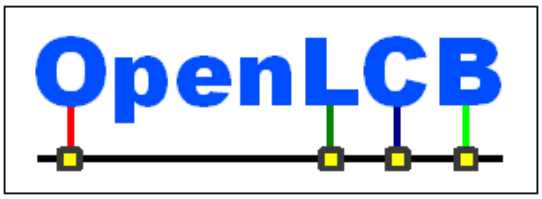

OpenLCB Technical Note	
Common Information	
~~Feb 6, 2016~~Feb 17, 2015	~~Adopted~~Adopted

1 Introduction

The OpenLCB Standards are independently normative. That is, each document contains statements that assert how things should or ought to be, and each document is written to be self consistent. In general, documents marked as Standards focus on explicit requirements and assertions, while Technical Notes provide non-binding context and rationale; to avoid confusion, the terms Normative and Informative are liberally used in section headings.

This Technical Note provides common background information that may 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

2.1 Reserved fields

When **sending** a message, set reserved fields to zero value unless otherwise specified. When **transporting** a message, transport reserved quantities unchanged.

When **receiving** a message, consult the Standard: reserved fields come in two flavors. One is "reserved, send as zero, **ignore upon receipt**". This means that a receiving node shall process the message, ignoring the value in the reserved field (or assuming it was the default value of zero). Some future extension or additional information may not be available, but the interaction can be continued without it. The other flavor is "reserved, send as zero, **check upon receipt**". This means that if the receiving node finds a non-zero value in that field, it must abort processing the message, not perform any action requested by it, and if appropriate, should respond with an error code, preferably some form of "unimplemented". Future versions of the standard may define the bits marked with "check upon receipt" in a way that is incompatible with previous versions of the standard, including giving the rest of the message a completely different meaning when the reserved field is not the default value.

The zero value sometimes indicates a non-initialized value.

2.2 Numerical representation

Unless otherwise specified, OpenLCB fields are unsigned.

OpenLCB does not define a floating point representation. IEEE half-size (16 bit) floats are used in e.g. the Throttle protocols under development.

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

2.4 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.

2.5 Strings

OpenLCB strings are sequences of UTF-8 values[1]. This allows OpenLCB devices to represent all the international writing systems that are described by Unicode[2] with only a small processing burden in the most common (Western alphabet) case.

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.

Certain standards retain the 0x80 bit in the first byte as a way of eventually indicating other codings, particularly compression of the UTF-8 text. If this is possible, it's specified in the relevant standard.

---

[1]For an introduction to UTF-8 coding and how it represents international character sets, see e.g. http://en.wikipedia.org/wiki/UTF-8

[2]From the glossary of the Unicode standard, http://www.unicode.org/versions/Unicode6.1.0/ch03.pdf#G7404 : “UTF-8: A multibyte encoding for text that represents each Unicode character with 1 to 4 bytes, and which is backward-compatible with ASCII. UTF-8 is the predominant form of Unicode in web pages.” See sections 3.9 and 3.10 in the Unicode specification, link above.

For example, XML CDI strings can start with either the UTF-8 text for "<?xml" which starts with 0x3C, or with a 0x80 followed by an indication of the format of the compressed UTF-8 text that follows. See the individual standards for more information.

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 for specific quantities, 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". General byte sequences that don't represent a single value should be in spaced-hex format, e.g. "01 02 AB DE 00 00". 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 an OpenLCB message

OpenLCB messages are described in this common format:

Name	Description	Simple Node	Dest ID	Event ID	Common MTI	Data Content
	(optional field)	N	N	N	0x0000	

- Name: Standard name of the message.
- Description: Optional field, generally used if there is variant forms.
- Simple Node: Whether the message is included in the "simple node subset" (defined in the Message Networking TN).
- Dest ID: Whether this message includes a destination address for a specific node. If not, the message is global.
- Event ID: Whether this message includes an Event ID.
  Common MTI: The 16-bit value, typically in hex notation, of the full Message Type Indicator.
- Data Content: Summary description of data bytes, if any, included after fixed fields such as MTI and destination address.

Adaptation to CAN messages are described in this format:

Name	Description	CAN-MTI	CAN Header	Data Content
	(optional field)	0x000 (optional field)	0x1800,0sss	fddd

- Name: Standard name of the message.
- Description: Optional field, generally used if there is variant forms.
- CAN-MTI format: Typical CAN frame representation of the MTI. Optional field, used only if the CAN-MTI is not equal to the low 12 bits of the Common MTI.
- CAN Header: Contains the CAN header. Source node ID alias is represented by "sss". Destination node ID alias, if present, is represented by "ddd".
- Data Content: Summary description of data bytes, if any. For addressed messages, also include the fixed destination address format, represented as "fddd", where the "f" represents fragmentation flags.

Note that this information is not completely independent. The MTI value depends on simple subset, priority subgroup, etc, values. The messages are detailed in specific associated documentation. That documentation is considered normative.

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

Table of Contents

