OpenLCB Technical Note	
Method for Allocated MTI Values	
Jul 30, 2011	Preliminary

1 Introduction

OpenLCB messages carry a Message Type Indicator (MTI) field to indicate their type. The primary form is a 16-bit number.

This note describes how the numeric values for those MTIs are allocated. The current allocations are documented in a separate spreadsheet[1]. Those allocations are normative. The discussion in this note is not normative on OpenLCB users or node developers, but does describe the methods that are to be used for allocating new MTI values for new OpenLCB message and protocol types.

MTI information may be carried in different formats for different wire protocols. These are documented in the same spreadsheet, and discussed in separate sections below.

2 Common MTI Values

Many nodes will treat MTIs as magic 16-bit numbers, just comparing them for equality to specific values of interest. That's a perfectly fine implementation strategy.

To allocate them, however, having a systematic pattern to the values can be more useful.

- The two most-significant bits are reserved as 00; nodes must send and check that value.
- The next bit is used to indicate unaddressed messages meant for "simple" or minimal nodes. A 1 in this bit means that these simple nodes can ignore this message. A 0 in this bit means that the simple nodes should process the message.
  This is meant to speed processing of some message types. The decision as to whether a particular message type is of interest to the notional "simple" node is left to the actual protocol definition, found elsewhere. This bit is reserved to 1 for addressed messages, as a message specifically delivered to a node should be processed.
- The next bit, the least significant of the top nibble, is reserved as 1; nodes must send and check that value.
- The next eight bits form a specific message number. This also has some substructure:
  - The most significant bit is reserved as 0; nodes must send and check that value.
  - The next 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

[1]See http://openlcb.org/specs/index.html for the current version of the spreadsheet. It provides concrete examples that may help you understand the material in this document.

of these two. Priority processing is permitted but not required. The priority group bits are part of the overall message type.

  - The next bit is reserved as 0; nodes must send and check that value.
  - The next 4 bits, forming the low nibble, are used to indicate the specific message within a priority group.

- The bottom nibble is interpreted as flags that define the structure and format of the message type.
  - The most significant bit is reserved as 0; nodes much 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 at a specific location in the message content.
  - 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 is at a specific location in the message content.
  - 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 for some wire protocols, see e.g. the definition of the CAN wire protocol.

  These flags allow nodes to do simple decoding of messages with MTIs that they don't recognize, perhaps because they were defined after the node was created.

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.

MTI layout:

Field	Reserved	Simple Node flag	Reserved	Message Number	Reserved	Dest ID flag	Event ID flag	Flag Byte flag
Size & location	2 bits 0xC000	1 bit 0x2000	1 bit 0x1000	8 bits 0x0FF0	1 bit 0x0008	1 bit 0x0004	1 bit 0x0002	1 bit 0x0001
Value(s)	0 (send and check)	0 means for simple node, 1 means simple node can skip	1 (send and check)	See below	0 (send and check)	1 means Destination ID present, 0 means not present	1 means Event ID present, 0 means not present	0 means flag byte present, 1 means not present

Message Number byte (8 bits) layout:

Field	Reserved	Static priority groups	Reserved (send and check)	Specific type
Size & location (within 8-bit field)	1 bit 0x80	2 bits 0x60	1 bit 0x10	4 bits 0x0F
Value(s)	0 (send and check)	0 to 3 0 goes first, 3 last if priority processing is present	0 (send and check)	See spreadsheet for values & meanings

Specific values are allocated and documented in the separate worksheet. We keep them in just that one place to avoid conflicting updates.

3 CAN MTI Considerations

The standard CAN MTI field is 12 bits in the header (messages without destination address) or one byte in the data segment (addressed messages). Since CAN frames only carry 8 data bytes, a 1-byte MTI short form will be used until future expansion makes more necessary. The possibility of longer MTI values has been reserved.
MTI information is carried in a different format on CAN links to increase bandwidth efficiency, simplify decoding in small processors, and permit use of hardware filtering.

Standard MTIs are mapped to one of seven types:

Type	**Meaning**
0	Unaddressed MTI for simple nodes
1	Unaddressed MTI for non-simple nodes
2	(Reserved, not used)
3	(Reserved, not used)
4	Datagram not-last segment
5	Datagram last segment

Type	Meaning
6	Addressed MTI other than datagram or stream
7	Stream Data

The bit coding of this has been chosen to use the "4" bit to indicate that a destination address is present, directly mapping to the "Destination ID present" bit in the full MTI.

The 4, 5 (Datagram) and 7 (Stream Data) values are special cases chosen for efficient processing of large amounts of data on CAN. Most MTIs will map to 0, 1 or 6.

The 0 vs 1 values map directly to the "Simple Node flag" in the full MTI.

3.1 Addressed CAN MTIs

Common MTIs with the "destination ID present" bit set are mapped to and from type 6.

In this case, the 12-bit destination alias is placed in the header, and the Message Number byte from the full MTI is the first byte of the CAN frame.

3.2 Unaddressed CAN MTIs

Common MTIs with the "destination ID present" bit set are mapped to and from type 0 or 1, depending on whether the "simple node" bit is reset or set.

In this case, the next twelve bits of the CAN header are available for MTI information. The first (most significant) eight bits are used for the Message Number byte. The last (least significant) four bits are used for the Event ID present, the Flag Byte present flag, and any flag bits themselves.

4 (Information to be integrated)

The following has been cut&pasted from other, older notes, and needs to be integrated with the main TN text above. This is a work in progress!

4.1 Bits vs Codes

Should a common Message Type ID contain bit-coded fields for e.g. presence of Destination Node ID, presence of Event ID? There are arguments in both directions.

Having a bit-coded value (bit that says Destination Node ID present) is the same as a bit saying "global message vs addressed"; one could chose to document it either way. Having this bit saves space in the most common messages, which are the global P/C Event Reports.

The alternative is a global address value. One could _always_ have the DID present, and have a specific value (e.g. 0) that indicates the message is addressed to all nodes. This may result in simpler message deformating code, as the remaining fields are always in the same byte locations.

The common message can be reformatted on any particular wire protocol, so if one form or the other works better e.g. on a CAN link, the local format can use the other form.

Similarly, having a bit that indicates an Event ID is present can help parse messages faster, but it's truely optional in any particular local format.

Table of Contents