



## OpenLCB Technical Note

### Event Identifiers

Mar 30, 2023

Draft

## 1 Introduction

This technical note contains informative discussion and background for the corresponding “OpenLCB Event Identifiers Standard”. This explanation is not normative in any way.

## 2 Annotations to the Standard

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

### 2.1 Introduction

- 10 Although event identifiers generally take the format of a Unique Node Identifier plus a 16-bit extension, there are some exceptions to this noted in the Event Identifiers Standard which will be detailed in this document.

### 2.2 Intended Use

- 15 An Event Identifier is intended to be globally unique and for a specific purpose defined either by user configuration or enforced by the Event Identifiers Standard. Any node may be configured to produce or consume any event, regardless of its assigned Node ID. However, logically, a unique Event Identifier is defined to have a unique purpose, whether assigned by user configuration or explicitly defined in the Event Identifier Standard.

- 20 The "globally unique" requirement only refers to the universe of connected nodes; nodes that never need to communicate with each other don't need to have separate Event Identifiers. In general, however, nodes can move: they can be sold or loaned for use on another layout, nodes on modular layouts can be connected to other arbitrary modules, and few assumptions can be made. Therefore, we require global uniqueness for all Event Identifiers.

To ensure uniqueness, the top six bytes of an Event Identifier that the manufacturer or user defines are required to be within a Node ID space that the manufacturer or user controls. The low two bytes can be any number, so long as each value is used for only one event.

- 25 This requirement applies equally to events defined by a hardware node, like a push button, a software node in a computer, or Event Identifiers that are defined by a human writing them on a piece of paper. In each case, the thing doing the definition must ensure it has control over the Node ID corresponding to the top six bytes, so it can ensure that the Event Identifier not be reused.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Node ID assigned to you						Unique 16 bit extension	

30

The Node ID part can be from real nodes which a user owns.

A software configuration tool might define events and assign Event Identifiers to them. A board manufacturer may prefer a pushbutton configuration process. A modular club may decide that certain events form the “boundaries” of modules, and need to be assigned Well-Known Event Identifiers to make it easier to create large modular layouts. Whatever the method, the Event Identifiers need to be globally unique, which is ensured by requiring Event Identifiers to be created using Node ID numbers assigned to them (and therefore not assigned to anybody else), plus an additional 16 bits that they are responsible for using only once. This is worth repeating, each Event Identifier can only be used once for a specific state or meaning. It doesn't mean that multiple nodes cannot use it, but rather it should not be used for a different purpose or meaning. This is because the meaning and its use is shared across all of the nodes using it, and unless it can be guaranteed that they all have been changed there will be conflict between them. It is much safer to use a new Event Identifier, and nodes will usually have a mechanism to supply new, “virgin” Event Identifiers.

Note that the Node ID part of an Event Identifier does not have to correspond to any physical node. So long as it is assigned out of a Node ID address space that the assigning body owns, and, by extension, has control of over, it can be used. For example, a fast-clock manufacturer might want to define a range of Event Identifiers so that a different event is emitted every fast minute. For the sake of argument, let's say 24 bits worth of specific events are needed (it's actually smaller, but let's use this as an example). The manufacturer has a large range of Node IDs assigned already, so can use 8-bits worth of that space, plus the extra 16 bits in the Event Identifiers below the Node ID portion, to do this. If a manufacturer had a range of Node IDs that included at least 12.34.56.78.9A.00 through 12.34.56.78.9A.FF (in other words, all possible values of the low byte), they could create a set of Event Identifiers with the low 24 bits used to carry a time value:

Table 2: Example of defining a Set of Events

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Assigned Node ID						Unique 16 bit extension	
0x12	0x34	0x56	0x78	0x9A	Time Value		

55

This is guaranteed to be unique, no matter where one of these devices is used, because the node ID range is guaranteed to belong to the manufacturer, and they will use it only once for this purpose.

2.3 References and Context

This Standard is in the context of the following OpenLCB Standards:

- The CAN Physical Layer Standard, which specifies the physical layer for transporting OpenLCB-CAN frames.
- The Message Network Standard, which defines the basic messages and how they interact. Higher-level protocols are based on this message network, but are defined elsewhere.

- The Event Transport Standard, which defines the protocol for transporting events.
- 65    • The Unique Identifiers Standard which defines the format and allocation of unique 48-bit identifiers.

## 2.4 Format

Event Identifiers are intended to be globally unique 64-bit values.

## 2.5 Allocation

### 70    2.5.1 Node ID Based

The majority of Event Identifiers fall into this category. The six most significant bytes are derived from a globally unique six byte Node ID followed by the two least significant bytes which are typically chosen through user configuration.

#### 2.5.1.1

### 75    2.5.2 Well-Known Automatically-Routed

Unlike Node ID Based Event Identifiers, the Well-Known Automatically Routed Event Identifiers are explicitly defined by the Event Identifiers Standard for a specific purpose. Additionally, gateways are required to route these events to all segments.

80    Note a change that it is not specified whether or not the Well-Known Automatically Routed events participate in producer/consumer identify. The primary purpose of the producer/consumer identify is to pair consumers with produces, especially for the purpose of routing between OpenLCB segments. Since these Well-Known Automatically-Routed events must always be routed, it could be argued that the identify events for these Well-Known Event Identifiers are redundant, but are retained for symmetry.

85    Though not explicitly required, it would be prudent for a manufacture to identify in a product's documentation as to which Well-Known Automatically Routed events it produces and consumes, and what are the resulting actions it takes when these events occur.

#### 2.5.2.1 Emergency Off

90    The Emergency Off Event Identifier (01.00.00.00.00.00.FF.FF) is a request for a node to de-energize all of its outputs. A node receiving this event may continue to remain a powered participant of the OpenLCB bus, but may de-energize any outputs unrelated to maintaining OpenLCB communications. The meaning of de-energize is not prescribed for any given node, it is up to the node manufacturer and/or user to prescribe what, if anything, should happen in the node if it receives this event. For  
95    example:

- If the node is a DCC Power Station, it might disable its amplified DCC output.
- If the node is an accessory controlling turnout motors, it might remove power from the motors

A node may revert to its previous energized state following the reception of a Clear Emergency Off event (01.00.00.00.00.00.FF.FE).

- 100 A node that has recently joined the OpenLCB network is not expected to know about or react in any specifically prescribed way to the current Emergency Off status defined prior to the node joining the network until the next Emergency Off or Clear Emergency Off event is produced.

### 2.5.2.2 Emergency Stop

- 105 The Emergency Stop Event Identifier (01.00.00.00.00.00.FF.FD) is a request for a node to command all of its outputs to a safe state. A node receiving this event is not required to de-energize any of its outputs. The meaning of “safe state” is not prescribed for any given node, it is up to the node manufacturer and/or user to prescribe what, if anything, should happen in the node if it receives this event. For example:
- 110
- If the node is acting on behalf of one or more DCC trains, it might send the global emergency stop command onto the DCC signal bus.
  - If the node is an accessory controlling turnout motors, it might do nothing, route its outputs to a manufacture default state, route its outputs to a user defined state, or something else altogether.

- A node may revert to its previous non-Emergency Stop state following the reception of a Clear Emergency Stop event (01.00.00.00.00.00.FF.FC). A node that has recently joined the OpenLCB network is not expected to know about or react in any specifically prescribed way to the current Emergency Stop status defined prior to the node joining the network until the next Emergency Stop or Clear Emergency Stop event is produced.
- 115

### 2.5.2.3 Power Supply Brownout Detected

- 120 The Power Supply Brownout Detected (node) Event Identifier (01.00.00.00.00.00.FF.F1) may optionally be sent by a node in the case where the node detects a low voltage condition that it considers abnormal, or out of its normal operating range. This event is agnostic to any other OpenLCB standard, and is a generic indication that it does not have enough voltage to function properly, regardless as to where that voltage originates. The node is not required to know the exact voltage at which the condition was detected in order to send this event.
- 125

- The Power Supply Brownout Detected (standard) Event Identifier (01.00.00.00.00.00.FF.F0) may optionally be sent by a node in the case where the node detects a low voltage condition that crosses the threshold required by any relevant OpenLCB Standard. It is implied that the node producing this event is subject to, and subsequently detecting a violation of, one or more OpenLCB Standards. The CAN Physical Layer Standard requirement of nodes operating down to CAN bus voltages of 7.5V is one example.
- 130

The exact time at which these events are sent is not specified and left up to the node’s designer. A number of possible options might be considered, which are not necessarily mutually exclusive. For example:

- 135
- The event may be sent immediately. Because the node is in a brownout state, it is reasonable not to assume that the event actually gets sent successfully and observed by other nodes in the network.

- The condition which would have resulted in the event may be logged such that it can be sent at a later time when the voltage has recovered to a point where successful transmission and observation by other nodes in the network is likely.
- The condition which would have resulted in the event may be logged such that it can be sent on the next node initialization when the voltage has recovered to a point where successful transmission and observation by other nodes in the network is likely.

It is not specified how other nodes in the network are to act upon these two events or that these two events are distinguished in any way when interpreted and presented to a user. The intent is that these two events, when observed, are useful for diagnosing network problems.

It is not specified how any given node will produce or inhibit production of these events when reacting to otherwise normal inrush conditions which may temporarily result in a low voltage condition which the node is designed to tolerate and recover from. The trade-off between false positive and false negative detection of brownout conditions is left up to an individual node's designer(s).

#### **2.5.2.4 Other Well-Known Automatically Routed**

Other Well-Known Automatically Routed Event Identifiers not discussed here have their uses prescribed and/or discussed elsewhere. No assumptions should be made about the use of these Well-Known Automatically Routed Event Identifiers as prescribed by this document.

#### **2.5.3 Well-Known**

Unlike Node ID Based Event Identifiers, the Well-Known Event Identifiers are explicitly defined by the Event Identifiers Standard for a specific purpose. Gateways are not required to actively route these events to all segments, and may maintain a static or learned routing table for these events to prevent unnecessary propagation.

Though not explicitly required, it would be prudent for a manufacture to identify in a product's documentation as to which Well-Known events it produces and consumes and what are the resulting actions it takes when these events occur.

##### **2.5.3.1 Duplicate Node ID**

The Duplicate Node ID Detected event would typically be sent by a node that receives a packet from another node containing its own unique Node ID. The production of this event by a node is not explicitly required.

##### **2.5.3.2 MERG CBUS**

OpenLCB allocates a Node ID address space specifically for mapping MERG CBUS events into OpenLCB. CBUS events come in two types: long and short. A long event is uniquely identified by the CBUS Node ID and Event ID. A CBUS short event is translated into an OpenLCB event by using 0x0000 as the Node ID in the "CBUS Node ID" field of the OpenLCB event identifier. This is allowed because in the CBUS standard, node-id 0x0000 is not permitted.

CBUS events can also come in the form of a request. These requests should be translated into an Identify Producer on OpenLCB in order to solicit the response that the CBUS request is asking for.

175 Two Identify Producer messages will have to be sent, one for the ON state range and another for an OFF state range, the response of which will need to be translated back for CBUS.

**2.5.3.3 Basic DCC Accessory Decoder**

180 The Well-Known Event ID space for basic DCC accessory decoders is provided as a suggestion to manufactures on how to translate OpenLCB events to classic DCC accessories. It is possible that there could be future mapping schemes required to implement a use case not covered by this scheme. This scheme is not designed to be mutually exclusive of other possible schemes, and its use is not required when mapping between OpenLCB events and classic DCC accessories.

The format of the DCC packet for a basic DCC Accessory Decoder is as follows (most significant bit first):

Byte 1	Byte 2	Byte 3 (Error Byte)
0b10aaaaaa	0b1AAACDDD	0bEEEEEEEE
Bits 5 down to 0 are the middle 6 bits of the address.	Bits 6 down to 4 are the 3 most significant bits of the address in 1's complement form. Bits 2 down to 0 are the 3 least significant bits of the address. Bit 3 is set for active, clear for inactive.	

185 The Accessory Decoder address format in most significant bit first format is: 0bAAAaaaaaaDDD, and it is expected that this is how the bits map to Bytes 7 and 8 of the Event Identifier. Though the AAA address bits are in 1's complement form under the DCC standards, they are in standard non 1's complement form in OpenLCB format.

190 By convention, most DCC Accessory Decoders couple two outputs together to form an output pair, and use the least significant address bit to select the direction of travel for the output pair as a whole. In this mode, the active/inactive bit 'C' is always sent as set or ignored altogether by the DCC decoder. The effective address space is cut in half (max of 2048 addresses); one set of four addresses are used for broadcast space; and another set of four addresses are thrown away by convention due to indexing starting at one. This is where the available 1-2040 accessory decoder addresses that most DCC users are familiar with comes from.

200 There is some ambiguity in how the 1-2040 addresses that most DCC users are familiar with translate to the bit level addresses between existing DCC systems in the market place. There are generally two schemes that are deployed, as follows in bold/underline with a repeating pattern throughout the address space. It is not prescribed how the OpenLCB user interfaces map to the bit level events. Grey cells mark addresses that do not exist in the opposing scheme.

User Facing Address	DCC Vendor A Binary Address		DCC Vendor B Binary Address	
	DCC Format	OpenLCB Format	DCC Format	OpenLCB Format
Reference	0bAAAaaaaaaaDDD	0bAAAaaaaaaaDDD	0bAAAaaaaaaaDDD	0bAAAaaaaaaaDDD
1 (normal)	0b1110000001001	0b0000000001001	0b1110000001001	0b0000000001001
1 (reverse)	0b1110000001000	0b0000000001000	0b1110000001000	0b0000000001000
252	0x111111111111x	0b000111111111x	0b111111111111x	0b000111111111x
<b>253</b>	0b11 <u>1</u> 00000000x	0b00 <u>0</u> 00000000x	0b11 <u>0</u> 00000000x	0b00 <u>1</u> 00000000x
<b>256</b>	0b11 <u>1</u> 000000011x	0b00 <u>0</u> 000000011x	0b11 <u>0</u> 000000011x	0b00 <u>1</u> 000000011x
257	0b110000000100x	0b001000000100x	0b110000000100x	0b001000000100x
508	0x110111111111x	0b001111111111x	0b110111111111x	0b001111111111x
<b>509</b>	0b1 <u>10</u> 00000000x	0b00 <u>01</u> 00000000x	0b1 <u>01</u> 00000000x	0b0 <u>10</u> 00000000x
<b>512</b>	0b1 <u>10</u> 000000011x	0b00 <u>01</u> 000000011x	0b1 <u>01</u> 000000011x	0b0 <u>10</u> 000000011x
513	0b101000000100x	0b010000000100x	0b101000000100x	0b010000000100x
1788	0x001111111111x	0b110111111111x	0b001111111111x	0b110111111111x
<b>1789</b>	0b00 <u>1</u> 00000000x	0b11 <u>0</u> 00000000x	0b00 <u>0</u> 00000000x	0b11 <u>1</u> 00000000x
<b>1792</b>	0b00 <u>1</u> 000000011x	0b11 <u>0</u> 000000011x	0b00 <u>0</u> 000000011x	0b11 <u>1</u> 000000011x
1793	0b000000000100x	0b111000000100x	0b000000000100x	0b111000000100x
2039	0b00011111000x	0b11111111000x	0b00011111000x	0b11111111000x
2040	0b00011111011x	0b11111111011x	0b00011111011x	0b11111111011x
Broadcast	0b00011111xxx	0b11111111xxx	0b00011111xxx	0b11111111xxx

#### 2.5.3.4 **AdvancedExtended** DCC Accessory Decoder

~~There are no known advanced DCC Accessory Decoders currently for sale in the market. The DCC standards are somewhat ambiguous about the address bit order. The address space is reserved for the~~

future should the use of advanced DCC Accessory Decoders become better defined and more prevalent. The format of the DCC packet for an extended DCC Accessory Decoder is as follows (most significant bit first):

<u>Byte 1</u>	<u>Byte 2</u>	<u>Byte 3</u>	<u>Byte 4 (Error Byte)</u>
<u>0b10A<sub>7</sub>A<sub>6</sub>A<sub>5</sub>A<sub>4</sub>A<sub>3</sub>A<sub>2</sub></u>	<u>0b0<math>\bar{A}_{10}\bar{A}_9\bar{A}_80A_1A_01</math></u>	<u>0bXXXXXXXX</u>	<u>0bEEEEEEEE</u>
<u>Bits 5 down to 0 are the middle 6 bits of the address.</u>	<u>Bits 6 down to 4 are the 3 most significant bits of the address in 1's complement form. Bits 2 down to 1 are the 2 least significant bits of the address.</u>	<u>8-bit command.</u>	

210 The Extended Accessory Decoder address format in most significant bit first format is: 0bA<sub>10</sub>A<sub>9</sub>A<sub>8</sub>A<sub>7</sub>A<sub>6</sub>A<sub>5</sub>A<sub>4</sub>A<sub>3</sub>A<sub>2</sub>A<sub>1</sub>A<sub>0</sub>, and it is expected that this is how the bits map to Bytes 6 and 7 of the Event Identifier. Though the A<sub>10</sub>A<sub>9</sub>A<sub>8</sub> address bits are in 1's complement form under the DCC standards, they are in standard non 1's complement form in OpenLCB format.

215 The reason that the 11-bit extended DCC accessory address falls into Bytes 6 and 7, inconsistent with the use of Bytes 7 and 8 in the basic DCC accessory address, is so that a single producer/consumer identified range message can be used without violating the 50% space utilization rule.

Example Event to DCC packet mappings:

<u>User Address</u>	<u>OpenLCB Event ID</u>	<u>DCC Packet</u>
<u>1</u>	<u>01.01.02.00.01.00.04.XX</u>	<u>0b10000001 0b01110001 0bXXXXXXXX 0bEEEEEEEE</u>
<u>2</u>	<u>01.01.02.00.01.00.05.XX</u>	<u>0b10000001 0b01110011 0bXXXXXXXX 0bEEEEEEEE</u>
<u>3</u>	<u>01.01.02.00.01.00.06.XX</u>	<u>0b10000001 0b01110101 0bXXXXXXXX 0bEEEEEEEE</u>
<u>4</u>	<u>01.01.02.00.01.00.07.XX</u>	<u>0b10000001 0b01110111 0bXXXXXXXX 0bEEEEEEEE</u>
<u>2043</u>	<u>01.01.02.00.01.07.FE.XX</u>	<u>0b10111111 0b00000101 0bXXXXXXXX 0bEEEEEEEE</u>
<u>E-Stop</u>	<u>01.01.02.00.01.07.FF.00</u>	<u>0b10111111 0b00000111 0b00000000 0bEEEEEEEE</u>

### **2.5.3.5 Other Well-Known**

- 220 Other Well-Known Event Identifiers not discussed here have their uses prescribed and/or discussed elsewhere. No assumptions should be made about the use of these Well-Known Event Identifiers as prescribed by this document.

DRAFT

## Table of Contents

1	Introduction.....	1
2	Annotations to the Standard.....	1
2.1	Introduction.....	1
2.2	Intended Use.....	1
2.3	References and Context.....	2
2.4	Format.....	3
2.5	Allocation.....	3
2.5.1	Node ID Based.....	3
2.5.2	Well-Known Automatically-Routed.....	3
2.5.2.1	Emergency Off.....	3
2.5.2.2	Emergency Stop.....	4
2.5.2.3	Power Supply Brownout Detected.....	4
2.5.2.4	Other Well-Known Automatically Routed.....	5
2.5.3	Well-Known.....	5
2.5.3.1	Duplicate Node ID.....	5
2.5.3.2	MERG CBUS.....	5
2.5.3.3	Basic DCC Accessory Decoder.....	6
2.5.3.4	Advanced DCC Accessory Decoder.....	7
2.5.3.5	Other Well-Known.....	8