OpenLCB Technical Note	
OpenLCB Event Transport	
Jan 26, 2013	Preliminary

1 Introduction

This Technical Note contains informative discussion and background for the corresponding “OpenLCB CAN Event Transport Standard”. This explanation is not normative in any way.

OpenLCB nodes respond to local inputs and state changes by emitting Producer/Consumer Event Report (PCER) messages. When they do this, they are acting as “Producers”. OpenLCB nodes receive these messages, and can act on them locally if desired. If they do so, they are acting as “Consumers”. Together, this is called the Producer-Consumer Model of system control. In this model a particular concept, or event, is the sum of the inputs that trigger the production of the PCER by the producer(s) and the action(s) taken by the consumer(s). Its main advantage is the knowledge independence it imparts between all of the nodes involved in an event. Expressed in more anthropomorphic terms, neither the producers nor consumers or a PCER are aware of each other. This adds flexibility to the system, since more producers or consumers of the PCER can be added to implement the associated event, without concern of side effects to the the other nodes involved in the event.[i][1,2]

The P/C Event ID included in a PCER message is a 8-byte unique identifier. The method for assigning these and ensuring their uniqueness is in the Event ID Standard and Technical Note.

There are many possible ways to use these 64 bits, some of which are discussed as examples below. In particular, OpenLCB does not require or enforce any particular partitioning of the eight bytes. More than one node may emit the same P/C Event ID (note that the Node ID of the emitting node is available separately in the message). More than one node may receive and act on a PCER message with a specific P/C Event ID. The only requirement is that the 8-byte quantity be unique to the event.

2 Annotations to the Standard

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

2.1 Introduction

[1]D. Miorandi and S. Vitturi, “Performance analysis of producer/consumer protocols over IEEE 802.11 wireless links”, *Proc. IEEE Workshop Factory Communication Systems (WFCS)*, 2004

[2]The Producer/Consumer Model and Control System Design ControlLogix 1999 https://cours.etsmtl.ca/gpa774/Cours/old-24-03-04/Documentations/Rockwell/articles/Producer_Consumer.html

2.2 Intended Use

A particular PCER may be produced in response to physical input or state change on a layout, for example a contact closure or the activation of a block occupancy detector, or it can be in response to a non-physical change such as the arrival of a certain time. The same PCER can be produced by several different state changes. That PCER may result in some action at consumers. The term 'event' includes the state changes, the PCER message, and the resulting actions, and represent a specific idea or concept, such as “Night has fallen”.

For example, the concept of an “Emergency stop” event might include the actions of turning off track power, turning on a fault-indicator, and sounding a buzzer. The PCER may be produced by any of: a panic button, a throttle button, an over-current detector, or an anti-collision system.

The aforementioned “Night has fallen” event might be triggered by a fast-clock time, a toggle switch, or a control program, and its actions might include illuminating street lights, house lights, and the dimming of overhead lighting.

In addition, ranges of PCERs can be used for specific purposes. For example, a fast clock's messages can be implemented as a range of PCERs, where the low order bytes are the time.

2.3 References and Context

For more information on format and presentation, see:

- OpenLCB Common Information Technical Note

For more information on MTI values and how they are laid out, see:

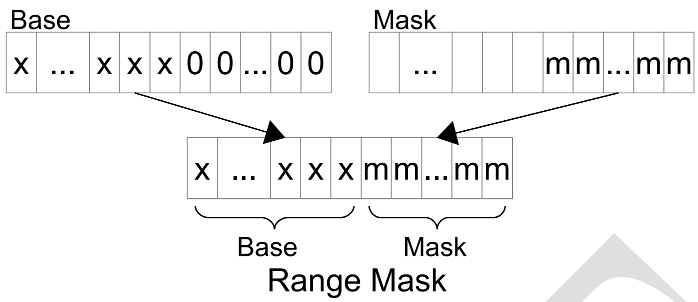
- OpenLCB Message Network Standard[3]
- MTI allocation table[4]. ~~http://openlcb.org/trunk/specs/MtiAllocations.pdf~~

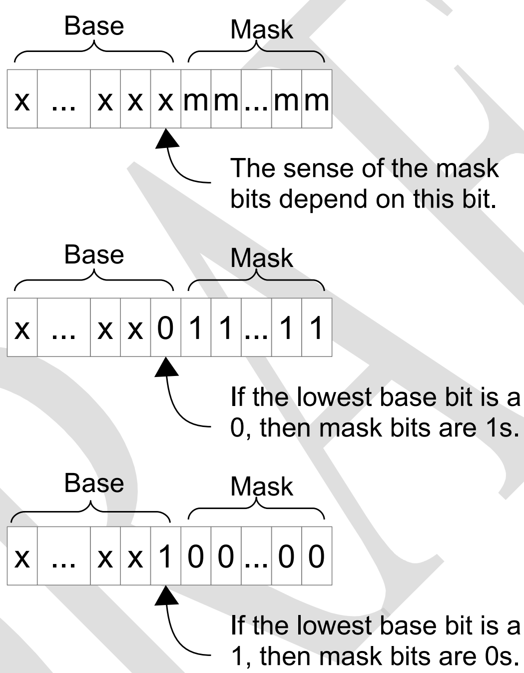
2.4 Message Formats

The range mask is implemented as a combined base value and mask in order to save space. The base value takes the form 0bv...0 and the mask 0b0...m. If the lowest bit of the base value is a one, then the combined form is 0bv...0. However, if the lowest bit of the base value is a zero, then the mask is inverted and the combined form becomes 0bv...1.

[3] As of the time of this writing, the OpenLCB Message Network Standard was still in preparation. See the OpenLCB web site documents page for current status: http://openlcb.org/trunk/specs/index.html

[4] http://openlcb.org/trunk/specs/MtiAllocations.pdf





Ranges can only be specified in powers of 2, ie they can specify ranges of 2, 4, 8, … .Therefore, if a range of n is required, then the next larger power of two is used. Ranges should not be over specified, so the standard requires that a range not extend past the next power of two by requiring that at least 50% of the EventIDs in the range be relevant. ~~and this specified in the standard.~~

Encoding a range is done by determining how many values are required, choosing the next higher power of two, choosing a base value~~,~~ and the mask, and finally combining the two, inverting the mask if necessary. For example, a range 0x1234xx would require a mask of the form 0x0000FF, and the combined form would be 0x1234FF. A range of 0x1235xx would also require a mask of 0x0000FF, but the combined form would be 0x123500, since the mask is inverted.

Decoding a range involves i~~n~~dentifying the mask as lowest contiguous bits of the same value as the lowest bit. The base value is the remaining high order bits. For example, the combined form 0x987650 would yield a mask of 0x00000F and a base of 0x98765, whereas 0x98764F would yield a mask of 0x00000F (inverted) and a base of 0x987640.

The range mask is implemented as a combined base value and mask, where the mask bits are the lowest n bits of the same value as the lowest bit. For example, the event-mask 0xABCD0000 would form the range 0xABCD0000-ABCDFFFF, since the lowest bit is a "0"and the mask bits extends to the first "1" bit, in this case the the low bit of the 'D' hex-digit. Similarly, the event-mask 0xFEDCBFFF would form the range 0xFEDC8000-FEDCBFFF, since the low-bit is a "1", and lowest "0" bit is in the 0xB = 0b1011, and at the third bit position. The smallest ranges are specified by 0xyyyyyyyyE, which specifies a range of 0xyyyyyyyE-0xyyyyyyyF, and 0xyyyyyyyD which specifies 0xyyyyyyyC-0xyyyyyyyF. 0X00000001 gives 0x00000000-0x00000001, and 0x00000002 gives 0x00000000-0x00000002, and 0x00000003 gives 0x00000000-0x00000003.

Event-Mask	Effective Mask	Bits	#	Resulting Range
0x00000001	0xFFFFFFFE	1	2	0x00000000-0x00000001
0x00000002	0xFFFFFFFE	1	2	0x00000002-0x00000003
0x00000003	0xFFFFFFFC	2	4	0x00000000-0x00000003
0x00000004	0xFFFFFFFC	2	4	0x00000004-0x00000007
0x00000005	0xFFFFFFFE	2	4	0x00000004-0x00000005
0x00ABCDEF	0xFFFFFFF0	4	16	0x00ABCDE0-0x00ABCDEF
0x00ABCDF0	0xFFFFFFF0	4	16	0x00ABCDF0-0x00ABCDFF
0x00ABCDE0	0xFFFFFFE0	5	32	0x00ABCDE0-0x00ABCDFF
0x0000FFFF	0xFFFF0000	16	65k	0x00000000-0x0000FFFF
0xABCD0000	0xFFFF0000	16	65k	0xABCD0000-0xABCDFF
0xFFFF0000	0xFFFF0000	16	65k	0xFFFF0000-0xFFFFFFFF

2.4.1 Producer/Consumer Event Report (PCER)

This message transports an Event-number from a producer node(s) to zero or more unspecified consumer nodes. It is the backbone of the protocol, and forms most of the expected traffic.

The source node ID allows identification of which node sent the event. This can be used for diagnostic and status purposes, but nodes must act on the event without regard to where it came from.

2.4.2 Identify Consumer

This message is broadcast and requests every node to report if they consume this Event ID.

2.4.3 Consumer Identified

This message is broadcast, in response to a received Identify Consumer message, from each node that consumes the included Event ID. Nodes also send this as part of their startup so that other nodes know they want to consume particular events.

This is one of the messages that allows bridges to do automatic routing of event messages.

The “valid”, “invalid”, “unknown” subforms are used to attempt to recover the state of the overall system. In general, event transmission via PCER messages conveys state changes. If a node comes in during operation, it might want to determine the correct value of the distributed state. Producers can, but don't always, know that state. For example, consider “On” and “Off” events for a light. There might be two producers attached to toggle switch. In that case, one of them will still see their state as current, and will reply with valid. Alternately, the two producers might be set to produce when two separate pushbuttons are pressed. In that case, at some later time neither producer knows whether the distributed state corresponds to its event. They both have to report as unknown. Consumers can also know the state. The lamp controller that consumes the “On” and “Off” probably knows whether it is currently providing power to the lamp, and so can reply with valid for one event and invalid for the other.

2.4.4 Consumer Range Identified

This message broadcasts, in response to a received Identify Consumer message, from each node that consumes event is in the range specified by the included event-ID-with-mask. This is one of the messages that allows bridges to do automatic routing of event messages.

The 50% is to allow rounding up to the enclosing bit, but no further.

2.4.5 Identify Producer

This message is broadcast and requests every node to report whether they produce this Event ID.

2.4.6 Producer Identified

This message is broadcast, in response to a received Identify Producer message, from each node that produces the included Event ID. This is one of the messages that allows bridges to do automatic routing of event messages.

2.4.7 Producer Range Identified

The 50% is to allow rounding up to the enclosing bit, but no further.

An example of a producer range would be a fast clock node that sends a range of events to indicate the time. Since there are 24*60*60 = 86,400 seconds in a day, it needs a range containing 86,400 events. That could be expressed in 17 bits, so if the range was to start with an Event ID of 0x12.34.56.78.00.00.00.00, the range could be expressed as 0x12.34.56.78.00.01.FF.FF using 1 bits to represent the mask. If the range had to start at 0x12.34.56.78.FF.FE.00.00 for some reason, 1 bits would not properly represent the mask because 0x12.34.56.78.FF.FF.FF.FF is actually a different range. Using a 0 bit for the mask would work, giving 0x12.34.56.78.FF.FE.00.00 as the representation.

This message is broadcast, in response to a received Identify Producer message, from each node that produces events in the range specified by the included event-ID-with-mask. This is one of the messages that allows bridges to do automatic routing of event messages.

2.4.8 Identify Events

Two forms of identify events; recommendation on uses for global form

In some cases, the node may want to query a specific node as to whether id produces that event, in these case the directed form is appropriate. In other cases, the node producing the Identify Event will not know if any node uses that event, and therefore the global form is the more appropriate choice. Be aware that since this latter form is transmitted throughout the network, it can be relatively expensive in terms of bandwidth.

2.5 States

After the IC message is sent, but before any Producer/Consumer Event Report messages are sent, each node must identify all events produced or consumed via zero or more Identify Consumers, Identify Consumed Range, Identify Producers and Identify Consumed Range messages. These are not required to be in any particular order.

2.6 Interactions

Nothing prevents extra Producer/Consumer Identified messages. Nothing prevents combining replies to multiple requests. This allows simplified implementations, for example setting a bit to indicate that a reply can be sent when time/priority is available.

You can send identify/identified messages for automatically-routed Event IDs. Identify Producer/Consumer for well-known Event IDs require response; there's no exemption for those.

The state of the system can sometimes be constructed from the received Producer/Consumer Identified messages and their “valid”, “invalid”, “unknown” subforms. It is possible that these will result in conflicting and/or ambiguous state.

Note that the Advertised/Not-Advertised state machine is reset by the Initialization Complete message, and not by any lower level link (CAN or other) state machine.

Delay in sending the PI/PRI or CI/CRI messages after IC is OK, but there's no delivery guarantee for this node's events until those have been sent.

When a node changes the events it manages, it still needs to emit the PI/PRI and/or CI/CRI messages. For example, reconfiguration could cause this. It can be handled by sending individual messages, or by resetting and sending them all as part of that process.

There is no way to indicate that a node is no longer interested in a particular Event ID. (Sending IC again says that all events are not interesting, though). This could be added, but it is much harder for gateways to decide that an Event ID is not interesting than that it is interesting, because they have to keep a list of all the node IDs that are interested,and back that off. Better to just let the set of interesting Event IDs grow until the layout or gateway is reset.

To ensure that event messages are properly routed, nodes must announce when they start to produce or consume messages. Specifically, they must do this at two times:

- When the node is first initialized, after the “Initialization Complete” message has been sent.
- When a configuration change has added another Event ID that can be produced or consumed.

To announce new Event IDs, the node must transmit a Producer Identified message for each P/C Event ID it can newly produce, and a Consumer Identified message for each P/C Event ID that it can newly consume.

2.6.1 Event Transfer

The last sentence is the key to gateway traffic reduction. See below.

2.6.2 Event Enquiry

Two forms of identify events; recommendation on uses for global form. This can be a huge load, and should only be used when needed.

The two forms of this message are sent to request that the specified nodes report all the events they produce or consume. These reports can be ether Identified messages specifying individual Event IDs or ranges of events. One form is an unaddressed message sent globally to all nodes, and the other is an addressed message to a specific node.

It is useful to be able to rapidly determine which, if any, P/C Event IDs that a particular node is listening for and that it can emit.

This can be used as a configuration diagnostic.

To determine which P/C Event IDs can be send by a particular node, the inquiring node sends an Identify Events message addressed to the target node.

The node must reply with a Producer Identified message for each P/C Event ID it can produce, and a Consumer Identified message for each P/C Event ID for which it is listening.

2.6.3 Producer Enquiry

It is useful to be able to determine which, if any, nodes are configured to possibly emit a specific P/C Event ID message and their current state.

This can be used as a configuration diagnostic, and as a way of building filtering and routing tables.

To determine which nodes can send a particular P/C Event ID, a node sends an Identify Producers message carrying the desired P/C Event ID. This is an unaddressed message addressed to all nodes.

All nodes that are listening for that P/C Event ID reply with a Producer Identified broadcast message.

The “valid” bit indicates that the node's internal condition is consistent with sending this P/C Event ID. For example, assume a node sends P/C Event ID 2 when the input goes active and P/C Event ID 4 when the input goes inactive. Then if the input was active and the node was asked about P/C Event ID 2, it would reply “valid”; if asked about P/C Event ID 4, it would reply “not valid”. Depending on the node's structure, it might not always be possible to set the “valid” bit with certainty, in which case the “unknown” bit must be set.

You can query a state when you come up with the request-id message. Conflicting states can happen, and have to be addressed. You might also want to enquire of the consumers if you don't get a definitive answer from the producers.

In regard to valid/invalid/unknown, it is always possible to send only unknown, or to ignore the three sub-forms, but a lot of capability is then lost.

2.6.4 Consumer Enquiry

It is useful to be able to determine which, if any, nodes are listening for a specific P/C Event ID message.

This can be used as a configuration diagnostic, and as a way of building routing tables.

To determine which nodes are listening for a particular P/C Event ID, a node sends an Identify Consumers message carrying the desired P/C Event ID. This is an unaddressed message processed by all nodes.

All nodes that are listening for that P/C Event ID reply with a Consumer Identified broadcast message.

The "valid" bit indicates that the node is currently in the state it would be if this message had been received last. For example, assume a node sets its output active for P/C Event ID 2, and inactive for a P/C Event ID 4. Then if the output was active and the node was asked about P/C Event ID 2, it would reply "valid"; if asked about P/C Event ID 4, it would reply "not valid". Depending on the node's structure, it might not always be possible to set the "valid" bit with certainty, in which case the "unknown" bit must be set.

3 Background information

This section is general information of interest to the reader, implementers, etc.

Performance & loading at node startup, layout startup particularly on CAN. Numbers and simulations.

The distributed-state model of a layout. "Events" broadcast state transitions, which individual producers and consumers use to change their knowledge of the distributed state.

Include some examples and figures from http://openlcb.org/trunk/documents/notes/ProducerConsumerModel.html ?

3.1 Gateways

Gateways can route PCERs only to segments with nodes expressing interest by processing the other event messages.

The automatically-routed event range has to be routed. This can be done by making a permanent entry in routing tables, or any other implementation method.

Merge the example in http://openlcb.org/trunk/documents/examples/CAN-TCP-CAN-TCP-example.html

3.2 Implementation hints

Buffering issues, particularly on CAN: a node has to be able to keep up with PCER messages that can arrive about every millisecond (125000 bps / 120 bits per message)

- Don't do long linear searches to match an incoming Event ID to internal structures.
- Instead, consider just decoding the Event ID and setting a "process as available" bit, and then doing further work outside the CAN receive loop.

There is no requirement that events be processed in any particular order. Multiple PCER messages with the same Event ID may result in multiple actions inside a consumer, or may not, depending on the details of the consumer. There's a whole range of ways that events could be handled. A consumer might always handle them in the order received, and with the inter-event time intervals as received, down to the microsecond. Or it might do them in-order on a time-available basis, and generally be able to do them in 50 milliseconds. Or it might do them in the order that lets it get done fastest. Or whatever. Similar discussion goes for repeated events: Does the hardware have to do something twice? That depends. In some cases, it's not even clear what it means. Specifying this is the node manufacturers job, because they have uses in mind, and they make nodes to meet those uses. OpenLCB just arranges to deliver events for them their processing. Node manufacturers should certainly specify the performance of their nodes. More knowledge is good, but the standard doesn't compel anything because it's hard to write a testable requirement. The range of possible behaviors is so wide. Similar discussion goes for repeated events: Does the hardware have to do something twice? That depends. In some cases, it's not even clear what it means. Specifying this is the node manufacturers job, because they have uses in mind, and they make nodes to meet those uses. OpenLCB just arranges to deliver events for them their processing.

The Standard doesn't make any statements about delivery performance because it's specific to the link-types, arrangement of OpenLCB nodes & links, etc that make up a specific OpenLCB network. In general, OpenLCB is going to provide sub-millisecond, millisecond, or perhaps even 10's of millisecond timing quality, for example. Nor can it guarantee absolute order of events. (If Node A and Node B emit events at the same time on different segments, there's no guarantee of when node C will receive them, nor in what order, nor that Node D will receive it at the same time & in the same order as Node C).

3.3 Well known events

There are a small number of cases where a globally-allocated and reserved Event ID will simplify operation. These “well-known Event ID numbers” can be used to e.g. advertise that a node can provide a specific capability, or to tell locomotive control hardware to stop all trains instantly, etc.

For these to be useful, they not only have to be unique (so there are no collisions that accidentally trigger reactions to them), but they must also be well-known. So we have created a central spreadsheet on which uses can be recorded. This will eventually provide a machine-accessible record.

All of these are assigned with a reserved ID in their upper six bytes to ensure uniqueness and simplify recognition.

Nodes using these must mention them when listing the event IDs they produce and consume. Gateways may filter on these, but are not required to. (For ease of implementation, a gateway may just pass all events with the common top 6 bytes).

3.4 History section

For monitoring purposes, it was proposed that a PCER message carry a sequence number which increments each time the associated P/C Event ID is sent. It is not at all clear how to do this across nodes, or even across Producers within a single node, as there may be more than one thing that can cause the same P/C Event ID to be sent by a single board;. We decided this would cause more problems than it solved, and omitted it.

There is not room in a CAN frame for both a destination NID and a EID. The protocol has therefore been constructed so that any message carrying an EID (and that isn't a datagram) is globally addressed.

Table of Contents

