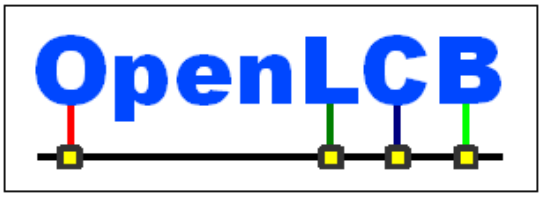

OpenLCB Working Note	
Location Service	
Oct 22, 2023	Preliminary

1 Introduction

A Working Note is an intermediate step in the documentation process. It gathers together the content from various informal development documents, discussions, etc into a single place. One or more Working Notes form the basic for the next step, which is one or more Standard/TechNote pairs.

This working note is the start of effort toward a general location service protocol which can be used by e.g. RFID and QR scanners and other detectors to publish information about the detected location of rolling stock.

Items in italics are basic open questions that should be addressed before this is published as a draft.

1.1 Served Use Cases

1) A signal needs to determine occupancy in the block it protects. It is not concerned with identity, but it does need to know location and entry/exit.

2) A train enters a current-detected block. A detector recognizes this, successfully queries the decoder for its address, and reports that information on the OpenLCB bus. That information results in a speed command such as Slow or Stop being sent to that particular train.

3) A train enters a current-detected block. A detector recognizes this, and attempts unsuccessfully to query the decoder for its address. The detector emits a report without decoder address information.

4) A train exits a current-detected block. A detector recognizes this, and reports that interaction on the OpenLCB bus.

5) A train crosses an RFID reader which determines the train's DCC address and reports that address on the OpenLCB bus.

6) A train-announcement unit needs to determine the identity and location of approaching trains so it can announce their arrival.

7) A train display needs to gather all train locations for display to the users.

8) A train display needs to gather the location and identity of all trains within a specified area.

9) An application needs to gather the location and identity of all trains.

10) A node powers up and emits a query which generates messages carrying all available location information.

11) An RFID reader reads a tag and uses this protocol to distribute the raw tag contents. A separate database node receives this content, does an internal lookup to locate the rolling stock's Unique ID, and then uses this protocol to distribute a human-readable version of the original distribution that also carries the rolling stock's Unique ID.

12) A detector observes that a piece of rolling stock has been added to the layout. It reports this. A throttle or command station provides that rolling stock in its roster display.

13) A detector observes that a piece of rolling stock has been removed from the layout. A throttle or command station removes that rolling stock from its roster display.

14) The user has a throttle, which has a train on it. There is a location report coming in for that train. The user would like to display on the throttle the name of the location where that train was seen. (This has to be a user-visible name, so coming out of some configuration.) This raises two indirection issues that need to be solvable:

    A) Indirection of the OpenLCB train and the detector address. Say this is a DCC train from a command station, then there is a known DCC address, which is recognized by RailCom detectors, and reported. The harder case of OpenLCB trains needs to be handled too.

    B) Indirection from the detector address to a user-visible (configured) block name.

15) Cab signaling: display on the throttle what the next signal aspect is for the selected train. This raises indirection issues:

    A) Going from train to block as above

    B) Going from physical orientation plus reverser setting on the logical locomotive to an absolute direction inside the block

    C) Going from block + absolute direction to a next signal (which is a multi-state state machine)

    D) Aspect interpretation (state to human readable text)

1.2 Unserved Use Cases

(To be determined as the proposal is developed)

1.3 Requirements

1) The protocol should cover multiple hardware types, including RFID and QR scanners, RailCom and other DCC detectors, and follow-on hardware that can detect and identify rolling stock at a point, on a track section, or in an area.

2) To the extent possible, the protocol should identify standard forms for common information.

    A) The Unique ID of the identified rolling stock should be in a common location in the protocol's message(s) so that it can be extracted regardless of the form of the sensor that generated it: a RFID detector and a QR detector on the same layout should both provide the same information on the identity of the detected rolling stock.

B) The Unique ID of the scanner should be in a common location in the protocol's message(s) so that it can be extracted regardless of the form of the sensor that generated it.

3) To the extent possible, the protocol should identify methods for its message(s) to carry detector-specific information. For example, a RFID detector and a QR detector on the same layout should both be able to publish their tag-specific information.

4) The identification of the scanning hardware should be guaranteed unique. This lets hardware be added to a layout without having to resolve conflicts.

5) The messages should not rely on the transmitting-node-address field of the message for decoding. I.e. the protocol doesn't say "You identify who made the scan by looking at who sent the message". This allows replacing the physical hardware that is broken with another piece of physical hardware carrying the same configuration; we expect that the entire network continues to operate as it did before. It is also useful to allow messages to be created by e.g. a debugging node during development.

6) It should be possible to look at the message(s) carrying a specific scan and know that they contain a scan without additional information. I.e. you don't have to configure a layout monitor with all possible scanner IDs before it can start processing or displaying scanning results.

7) The identification of which scanner is reporting should be early in the message(s). This allows nodes to drop the receipt of the message(s) early, without having to buffer them completely. (Note that the relative priority of this requirement and the one below has not yet been determined)

8) The identification of the detected rolling stock should be early in the message(s). This allows nodes to drop the receipt of the message(s) early, without having to buffer them completely. *(Note that the relative priority of this requirement and the one above has not yet been determined; this proposal ranks the previous one above this one)*

9) There should be a standard algorithm to find the UniqueID of the scanned piece of rolling stock within the message(s), when the scanner can provide one. In the common case of DCC, this is the UniqueID mapped from the DCC address. For some scanners, e.g some types of RFID scanner, the scanner might not be able to map the scanned value to a specific piece of rolling stock, in which case this information field would be blank.

10) Note that the item directly above is referring to a full Unique ID (node ID), not just a DCC address. Someday, there will be LCC-native trains and this protocol should be ready for them. The alternative is inevitably a messy migration at some point in the future.

11) Ideally, information about short circuit and current overload status could be conveyed by this protocol. These are often generated by the same device as the transponder messages e.g. LocoNet devices. These are providing location-specific information. It would be good to have some commonality between all location based messages. Another example would be current or voltage measurements at a point.

1.4 Not Requirements; Unmet Requirements

1. Ideally, the messages for this protocol should be a super-set of those from the DCC Detection Protocol. *(This does not seem possible now, but at an early stage of requirement generation one could dream...)*

2 Specified Sections

This is the usual section organization for a Technical Note, to accumulate the Standard and Technical Note content in its eventual order.

In this section, deeply indented content is intended for the eventual Technical Note.

2.1 Introduction

Note that this section of the Standard is informative, not normative.

A "scanner' is a device that can gather information from the railroad, typically when something happens. RFID readers, QR code readers, point optical detectors and track current detectors are all examples of scanners.

A scanner forwards one instance of the gathered information as a "report".

RailCom is a trademark of Lenz GMBH. Transponding is a trademark of Digitrax, Inc. StaRFIshrail is a trademark of 292 Design UK, Ltd.

2.2 Intended Use

Note that this section of the Standard is informative, not normative.

This protocol is intended to provide a standardized way for scanners to distribute report information across the OpenLCB network.

2.3 Reference and Context

See also the DCC Detection working note for a simpler form specific to DCC.

This working note assumes adoption of the Event With Payload working note as a Standard.

2.4 Message Formats

2.4.1 Report Message

The proposal is to carry reports within a Producer Consumer Event Report (PCER) as an event with payload defined by:

Note bytes 0-7 are the Event ID of the message, 8-264 are the payload of the message.

- Bytes 0-1: A defined prefix from the well-defined space to be assigned.
    - *0x0102 is recommended by this draft.*

      This unambiguously identifies this PCER as part of this protocol.

That enables nodes not interested in this protocol to discard this event-with-payload transmission immediately.

Using a unique prefix (e.g. the recommended 0x0102) followed by a UniqueID (see immediately below) is a mechanism for creating a fully-unique Event ID. No other Event ID allocation mechanism can duplicate one of these IDs.

- Bytes 2-7: A Unique ID that specifies the device that read the information e.g. a scanner.

  Having this in the 1st eight bytes, i.e. the Event ID, allows using the various identify message exchanges to locate scanners.

  Putting this near the start of the message (1st CAN frame) allows early rejection of uninteresting scanner sources.

- Bytes 8-9: Flag bits, defined below
- Bytes 10-15: A Unique ID that specifies the item being scanned, e.g. a train node ID, DCC address mapped to a Unique ID, or zeros if the information is not available.

  This will fit in the 2nd CAN frame, allowing early rejection of uninteresting items when available.

Is it the right choice to put the scanner ID in the Event ID for easy retrieval by the identify exchanges, while putting the scanned device in the content requiring more interactions to find them?

- Bytes 16-NN: Any read (scanned) content that's available.

  Note that the proposed 256 byte payload size allows 248 bytes here. The standard format (see below) uses at least two of those for headers, allowing a maximum of 246 data bytes.

2.4.1.1 Flag bits

bits 0-1: entry/exit

0b00: exit – the report refers to an item disappearing from the detected area

0b01: entry without state – the report refers to an item appearing in the detected area; no report of the item eventually disappearing from the detected area will follow

0b10: entry with state – the report refers to an item appearing in the detected area; a report of the item eventually disappearing from the detected area will follow

0b11: reserved, do not send; ignore field upon receipt

bits 2-3: direction of relative motion – defined with respect to the designated front of the piece of rolling stock.

0b00: stopped

0b01: forward

0b10: reverse

0b11: unknown

bits 4-5: direction of absolute motion – the geographical meanings of these are defined during set up of the scanner. Note that this is the direction of motion, not direction the front of the item is pointing.

0b00: stopped

0b01: east/north

0b10: west/south

0b11: unknown

bits 6-7: reserved, send as zero and ignore on receipt

Should this contain an indication of the direction the front of the item is facing? There could be scanners that can detect that but can't detect forward/reverse motion and east/west motion.

bits 8-11: reserved, send as zero and ignore on receipt

bits 12-15: content format – see below

The content format field is used to indicate the format of bytes 16-NN

0b00 reserved, do not send; ignore message upon receipt of 0b00

0b01 occupancy/location information only, no additional information present

Indicating "no additional information present", instead of there just being no additional bytes of information, could be reconsidered.

0b02 (initial) standard content form, see below

2.4.1.2 Standard Content Form

The standard content form consists of one or more blocks formatted as

byte 0: length of content N, including type byte and data but not including the length byte

Numerology: A byte is sufficient here, as the maximum length is restricted to 246 bytes by the Event Transfer protocol.

byte 1: type of content – see below

byte 2-N: content defined by type byte

There can be more than one block, in which case the blocks are concatenated with no intervening bytes. Blocks of length 0 (i.e. a single zero byte) can optionally appear at the end of the message.

2.4.1.3 Standard Content Block Types

The type-byte values are allocated by this document. It's expected that new types will be routinely added. The current set of types are:

0x00 reserved, do not send, ignore the associated block on receipt

This is meant as a convenient flag for software while creating and parsing messages.

0x01 human-readable format: not intended for machine parsing, this is meant to carry messages for human display: "Train 123 arriving at station ABC"

0x02 raw content of a read RFID tag.

Do we need to have subtypes here for the type of tag, i.e. ISO14443 vs ISO15693?

0x03 raw content of a read QR or bar code

0x04 raw content of a RailCom read

0x05 raw content of a Digitrax Transponding read

0x06 position information: Three 16-bit IEEE floating point numbers representing the X, Y and Z position of the detected object on the layout. The units are meters. The origin and direction of axes is specified when the scanner is configured by the user, but conventionally X is -left/+right, Y is -front/+back and Z is -down/+up from some location on the layout. This corresponds to putting the origin at the left-front of the layout on the lowest level.

Note that 8-bit IEEE does not have a suitable resolution at a suitable maximum range. 16-bit IEEE provides roughly 16mm steps at +/-25m

0x07 JSON formatted data

How is the specific JSON content defined? "Self-defining" is not great in a standard.

0x08 Analog information – for reporting analog information (voltage, current, temperature, water level, …) as an IEEE-16 value in SI units

How do we indicate what kind of information is being presented? We need something in the block so that we can have multiple analog blocks in the block. Note that there can even be multiple readings of a single type in the same message e.g. "Ambient temperature" and "Motor temperature"

1. *A single byte sub-type ID with a defined set of values and meanings*
2. *A null-terminated string that describes the type, with or without a defined set of possible values and their meanings.*
3. *A combination: a byte that identifies the physical quantity (volts, amps, degrees C, meters, radians, ...) with provision for an optional null-terminated string label ("Ambient" for a temperature reading)*

0x09 – 0xFF reserved, do not send; ignore the block on receipt

(Much more work needed here)

2.4.2 Consumed and Produced Event ID

Every node that produces information via this protocol also consumes a specific well-known Event ID consisting of the well-known prefix assigned above (0x0102 recommended) followed by six 0x00 bytes.

The six 0x00 bytes is an invalid Unique ID, so no node will allocate this as its own.

Nodes that will inquire as to the current values of reports produce a specific well-known Event ID consisting of the well-known prefix assigned above (0x0102 recommended) followed by six 0x00 bytes.

2.5 States

This protocol has no states.

Remembering that a report for entry to a detected area has been sent, so that an exit message can later be sent, is not considered a public state of the protocol.

2.6 Interactions

Note that the Unique ID that specified the device doing the reading (bytes 2-7) does not need to be the Node ID of the scanning node. It just needs to be uniquely assigned by the node creator from some assigned space to indicate the device doing the read.

For example, consider a board that supports four RFID readers that operate independently. Each of those readers will have a distinct Unique ID that is used when its detections are reported. In addition, the board will have a Unique ID that it uses as its node ID. This may, but does not have to be, the same as the Unique ID of one of the readers.

2.6.1 Reporting new information

When a scanner gathers new information, it may emit a PCER message with the above format. PCER messages are global, so all interested nodes will receive this message. The interested nodes can then process them as desired.

Note this says “may emit”. That choice of whether or not to report every time new information is received is up to the device manufacturer.

Note that a single node implementing multiple readers will send PCER event-with-payload messages from those separate readers but must not interleave the messages making up the payload of those separate PCER messages. The event with payload protocol requires that the sending node only sends one PCER event-with-payload sequence at a time.

Example: An RFID reader reads the fixed information from a tag. This does not directly contain a Unique ID for the associated item. The reader produces a report with a zero value for bytes 10-15, the identity of the scanned item, and a block of RFID information. A second node with a RFID → Node ID mapping database consumes that message,

looks up the appropriate Unique ID, and produces a report message that contains it and the original RFID information block.

Example: A scanner produces a message that carries only the minimal information. A second node with a database consumes that message,. Using the Unique IDs for the scanner and the scanned item, it creates suitable human-readable content such as “Train 123 arriving on track 4 five minutes late”. It then adds that content as a new block and produces a new message with otherwise the same content. Meanwhile, a node driving a station display consumes the first message, sees it’s from its associated scanner but there’s no human readable content and discards it. It then consumes the second message, sees it’s from its appropriate scanner and contains a human-readable block, extracts the human readable content, and displays it.

Example: A control node is tasked with controlling the speed of each train that approaches a signal. A scanner is configured to report the arrival and departure of each train on the track approaching the signal, including Unique ID and directional information. The control node recognizes reports from that scanner, extracts the train’s unique ID, and determines whether the train is approaching the signal or receding from it. The control node then has enough information to control the train via the traction protocol.

Example: A node facilitating cab signaling will consume all messages from this protocol. It consumes all report reporting arrival of all trains on the track approaching each signal. For each report, it determines if the train is approaching the signal, (somehow) determines the current aspect of the relevant signal, and then transmits that aspect to the relevant train throttle(s) using some protocol not defined here.

This could also be done on the throttle by checking for its own scanned ID in reports, but that requires substantial context information that seems better centralized.

Or there could be a mechanism for directly inquiring about the current aspect of the signal associated with a particular reader’s Unique ID, which would allow a distributed mechanism.

Example: A video system is directly tracking trains. For each train, it periodically produces a report that contains the X, Y, Z position of the train. Each of these comes from the same scanner ID, the Unique ID associated with the video system. For this to work with a signal system from another vendor, a node is provided that can compute occupancy in virtual blocks: When the train is inside a specific volume, that’s considered to be occupancy in a specific virtual block associated with a specific virtual scanner’s Unique ID. Upon finding a train in one of these, this node will produce a report that includes the Unique ID of the specific virtual scanner. This in turn will be consumed in the usual way by the signal system node(s) that are looking for occupancy on that associated section of track.

2.6.2 Identification of scanners

Sending the defined 0x01 02 00 00 00 00 00 00 Event ID in an Identify Consumer message will result in Consumer Identified messages that indicate which nodes can take part in this protocol as scanners.

This removes the need to include this protocol in the Protocol Identification Protocol bits, *although that can also be considered for ease of access.*

Once the nodes that use this protocol have been identified, an Identify Events directed to each of them will return Producer Identified messages indicating the Unique IDs of all the scanners.

Note that scanners are not individual full nodes. They're not expected to each implement the full OpenLCB protocol stack. Instead, this protocol has a single node providing that service for multiple scanners.

2.6.3 Identification of scannable objects

There is currently no way to efficiently request the unique IDs of all scannable objects. Some of them might not be currently visible to the system. You can retrieve all presently-visible objects by retrieving the most recent scan reports, see section below.

2.6.4 Retrieving most recent scan reports

When a new node joins the network, it may want to catch up on previous scan reports. In that case, the new node can produce (in a PCER message) the 0x01 02 00 00 00 00 00 00 Event ID. This is consumed by all nodes that implement this protocol. When consumed, each of those nodes will re-emit the most recent report from each of its scanner implementations.

How to handle multiple overlapping reports from the same scanner? Just send them all? In what order? Option for the implementor?

This might be a large load on the network. PCER messages are high priority, and this request will result on one or more messages from (most? all?) scanners.

2.6.5 Identifying users of scanned information

A node can identify all nodes that use scanned information by emitting an Identify Producer message for the 0x01 02 00 00 00 00 00 00 Event Id.

It works out this way, but is there any use for it?

The current language doesn't actually require that a using node be able to produce that event if it's not going to inquire about the current reports. So "identify all nodes that use scanned information" might be over-promising.

2.6.6 Traffic limiting

Producing nodes may not produce the same event with payload content twice in a row

This is a compromise. It reduces the amount of state that a producing node needs to retain down to one message regardless of the number of attached readers, while at the same time trying to reduce the bandwidth consumed by repeating detection messages.

Another approach would be to say that a node may not produce the same content twice in a row from each scanner that it supports. This will prevent "scanner A, scanner B, scanner A, scanner B" bursts of messages. The cost is additional state for each scanner.

Another approach would be to say that a node may not produce the same content more than N times a second. This may require more state information be retained than the other approaches.

3 Background Information

RFID readers and tags come in multiple forms. With 13.56MHz RFID, be it ISO14443 (as used in the RC522 readers), or ISO15693 as used in StaRFIshrail, the following points apply:

- Up to 8 bytes can be read "on the fly", i.e. as the tag on a train passes over a reader at speed. Depending on how the reader is set up, these 8 bytes can either be the UID of the tag, programmed in at time of manufacture, or 8 bytes of EPROM, which can be programmed into the tag by the user. The usage of these 8 bytes would be determined by the user, but could, say, be 2 bytes for Stock ID, 2 bytes for DCC address, and 4 bytes for other purposes.
- In addition, there at least another 96 bytes of user programmable EPROM, but these need to be read when the tag is essentially static over a reader, due to the time taken to read all the bytes.

The tags used by the MERG CAN RFID readers provide five bytes of programmable payload. Multiple conventions have been proposed for the contents of those bytes.

QR codes come in multiple forms. Depending on version, size and error correction level, they can contain thousands of characters.

Some scanners can report the direction of the detected train. For some, this is whether the train is moving in forward or reverse. For others, it's whether the train is moving in one direction on the track vs the other (North vs South, East vs West). ProTrak Grapevine systems can report moving North/East, not moving, and moving South/West.

RPS detectors reported absolute X, Y and Z coordinates of the detected rolling stock. The report was emitted at specified intervals. Although these are no longer being produced, something similar may emerge in the future with developments in e.g. video monitoring of a layout.

Traditional block detectors and point detectors are an interesting special case. They're simple, low-cost and common on layouts. They can't sense any address information from the detected rolling stock. The protocol should be able to convey their information too.

Table of Contents