OpenLCB Working Note	
Traction Protocol	
Apr 26, 2013	Preliminary

1 Introduction

This working note covers the Traction Protocol, the way that OpenLCB handles moving objects such as locomotives, engines, and other rolling stock.

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.

1.1 Terminology

"DCC" refers to NMRA DCC; "Legacy" refers to all pre-existing protocols including DCC, TMCC, Marklin, DCS, etc.

"Trains": For our purposes, Train is anything which can be independently controlled. In addition to a model of a prototype train from locomotive to caboose, it might be just single caboose, a set of lit & controlled passenger cars, a diesel MU lash up, or basically anything that can take an OpenLCB "decoder" or a DCC decoder with a legacy attachment.

"Throttles": For the purposes of discussion, we draw a distinction between three kinds of throttles that a user might encounter:

- "Legacy Throttles" refers to throttles designed for use with extant DCC systems, e.g. a Digitrax DT402 or Lenz LH100.
- "Full-Featured Throttles" refers to full-featured native OpenLCB throttles with multi-line color screens and effectively unlimited processing power, e.g. a software throttle implemented on an iPad.
- "Simple Throttles" refers to throttles which are native OpenLCB nodes like Full-Featured Throttles, but which have more limited capabilities, e.g. no text display, a limited array of physical buttons, and constrained processing resources.

"Proxies": In the long term, we expect that OpenLCB protocols will go all the way to the train. This has great advantages, because you're always in complete communication with the train, and don't have to worry about only being able to configure the train when it's on a service track, storing information somewhere else so that it can be retrieved while the train is moving, etc. But until radio or other technologies mature to the point that this is possible, "proxy nodes" can be used as stand-ins for that capability. A throttle might communicate with a node that's serving as a proxy for the train, handling the communications, keeping track of status & configuration, etc. Out the back end of that proxy node is some other kind of communications, perhaps direct DCC or a connection to a legacy system that in turn makes DCC signals, or some other technology entirely. Due to the nature of those back side communications methods, the proxy may not be able to do everything that OpenLCB can, or only do some of it at certain times. The OpenLCB traction protocols need to take this reality into account. Anther use case for proxies, discussed in detail below, is to provide the mechanism for consisting.

“Command Stations”: Existing DCC and other control systems use “command stations” to create a track signal for controlling the trains. Usually the command station is controlled from the user side by some other network, to which throttles and other interface devices are connected. OpenLCB, it it's native form, has no such concept. Devices, like throttles, that want to talk to a train do so directly. Only when working with legacy systems does the concept of a command station enter, and usually through the form of a proxy node that is acting for the Train.

“Consisting”: The running of multiple items together, e.g. three coupled engines, each with their own NodeID or DCC address, as a single locomotive. DCC systems provide this now in various ways and with various names.

1.2 Served Use Cases

1.2.1 Train Operation

Bill hasn't run his passenger train recently on his OpenLCB-equipped layout. He picks up a throttle, hits a few keys, sees his passenger train listed, selects it and starts to run it. Some configuration needs tweaking (e.g. volume too low), so he enters a configuration dialog on the throttle, finds the right item by reading through them, changes the value to be a few larger, and stores that back into the train as it's running on the main track. That makes it work immediately.

1.2.2 Large Modular Layout

Arnold has put his OpenLCB-equipped train on a large modular layout, where it is one of 500 pieces of equipment. He picks up a throttle, presses a few keys, sees his train, selects it and starts to operate it.

1.2.3 Train on New Layout

Jim takes his OpenLCB-equipped train to Bill's OpenLCB-equipped layout and puts it on the track. He picks up a throttle, hits a few keys, sees his train, selects it and starts to run it. On this layout, some configuration needs tweaking (e.g. volume too low), so he enters a configuration dialog on the throttle, finds the right item by reading through them, changes the value to be a few larger, and stores that back into the train. That makes it work. When he gets back home that value is still present so he changes it back using the same procedure.

1.2.4 Legacy Train on New Layout

Jim takes his DCC-equipped train to Bill's OpenLCB- and DCC-equipped layout and puts it on the track. He picks up a throttle, hits a few keys, sees his train, selects it and starts to run it.

As an alternative, Jim takes his DCC-equipped locomotive to the layout, puts it on the track, enters the DCC address into a throttle, and starts to run it.

1.3 Unserved Use Cases

1.3.1 Multiple Independent Command Stations

Large modular layouts use multiple command stations to increase the effective bandwidth of the DCC bus. This is not an explicitly supported use case in the current work. Future work may make this possible as an extension.

1.3.2 Improved Legacy Addressing

DCC systems cannot run two locomotives with the same DCC address at the same time. This will still be true when running DCC-equipped locomotives via OpenLCB protocols to the command station.

1.3.3 Third-Party Communications

Node A is a throttle that is controlling train node B. Node C passively listens to the traffic and reacts to throttle commands and train status by taking various actions, such as providing appropriate sounds or preventing the speed from getting too high.

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.

2.1 Introduction

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

2.2 Intended Use

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

2.3 Reference and Context

NMRA S9.2 and NMRA RP9.2.1 define the formats for DCC addresses

2.4 Message Formats

AA.AA refers to an NMRA short or long address in the format defined by the NMRA. (Say a few words about short addresses in two bytes, maybe give examples)

2.4.1 Defined Event IDs

IsTrain: 01.01.00.00.00.00.03.03

IsIdleProxy: 01.01.00.00.00.00.03.04

IsInUseProxy: 01.01.00.00.00.00.03.05

IsProxiedDccAddress: 06.01.00.CC.AA.AA.03.03

EmergencyStopAll: 01.01.00.00.00.00.FF.FF

2.4.2 Traction Control Command Message

MTI: Priority 1, index 15, modifier 2, addressed => MTI 0x05EA, CAN frame [195EAsss] fd dd

The MTI modifiers are chosen to have the reply a higher priority than the request, ensuring replies to repeated instructions are always possible. The same priority and index are used for command and reply messages, changing only the modifier, to use less of the high-priority MTI space which is a scarce resource.

This message type and MTI is specific to traction control. The first byte of the content codes an "instruction", which defines the rest of the format. The instruction codes were selected with the high nibble representing protocol (0x00 for OpenLCB full protocol; 0x80 for DCC legacy; others reserved)

Instruction	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set Speed/Direction	0x00	Speed and direction as signed float16						
Set Function	0x01	Address			Value			
Emergency Stop	0x02							
Query Speeds	0x10							
Query Function	0x11	Address						
Manage Proxy	0x80	Attach Node 0x01	Node ID					
		Attach DCC Address 0x81	DCC Address AA.AA		Speed steps (14, 28, 128)			
		Detach Node 0x02	Node ID					
		Detach DCC Address 0x82	DCC Address AA.AA					

Want to add "Query attached Node IDs" and "Query attached DCC addresses" as instructions.

2.4.3 Traction Control Reply Message

MTI: Priority 1, index 15, modifier 0, addressed => MTI 0x05E8, CAN frame [195E8sss] fd dd

Higher priority to ensure can be sent immediately over Traction Control Command messages. Coding and structure similar.

Instruction	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Query Speeds Reply	0x10	Set Speed		Status	Commanded Speed		Actual Speed		
Query Function Reply	0x11	Address			Value				
Manage Proxy Reply	0x80	Attach Node Reply 0x01	Node ID						Reply Code
		Attach DCC Address Reply 0x81	DCC Address AA.AA		Speed steps (14, 28, 128)	Reply Code			
		Detach Node Reply 0x02	Node ID						Reply Code
		Detach DCC Address Reply 0x82	DCC Address AA.AA		Reply Code				

The Query Function Reply is in the format of the Set Function message, with a different MTI and the query bit set.

Attach reply code: Zero or not present, OK

Detach reply code: Zero or not present, OK

Do we want to return the number of attached nodes/DCC addresses?

Error codes need to be defined.

- Allocation Failed, temporary (try again) handles race conditions *(but we need to look carefully at that race condition in practice: With the current structure, does it end up adding two DCC addresses to the single proxy, instead of reserving it? Probably need to separate out reservation-for-configuration. In general, two different things like "assign" and "reserve" should be handled by two separate things)*

- Allocation Failed, permanent (out of proxies, etc)

Want to add responses to "Query attached Node IDs" and "Query attached DCC addresses". These have unbounded length if the number of members in a consist is not bounded by the protocol. Could be a single large message, but everybody in the communication path still needs a maximum message size to ensure buffering. So a multiple message (or multiple datagram) response with an "last one" bit is the way to go. Using datagrams is more protocol work, but leverages existing protocol support and moves the response to a lower priority.

The Query Speed/Direction reply is almost in the Set Speed/Direction format, with the addition of the two additional speeds. If a node cannot provide any of those three speeds, it should use float16 NaN (not a number) 0xFFFF. "Set Speed" is the most recent speed received in a Set Speed/Direction instruction. "Commanded Speed" is the speed that the traction control is currently attempting to move, taking into account momentum and any other control modifiers. "Actual Speed" is the current measured speed of the locomotive. There is no accuracy guarantee for Actual Speed.

On CAN, the Query Speed/Direction reply does not fit in a single frame, so it's sent as two frames with start and end marked in the 1st data nibble (high part of destination address). The status byte was included so that the actual speed value would not be split across boundaries (though that's not necessarily guaranteed for other wire protocols that come along later) The status byte is reserved. Send 0x00, don't check on receipt. For example, from node with alias 123 to node with alias 456, all speeds equal to 0x4420 would be sent as the two frames:

195E8123 14 56 10 44 20 00 44 20

195E8123 24 56 44 20

2.5 States

Full OpenLCB nodes do not have any identified states for this protocol. (They do remember their speed, direction and functions) Emergency stop is not a state.

Proxy Node States:

- Idle – not allocated to a specific legacy address
- InUse – allocated to a specific legacy address and controlling the equipment (if any) at that address.

The IsInUseProxy and IsIdleProxy Event IDs are used to indicate transitions in the state of a proxy node. Proxy nodes are created in Idle state.

2.6 Interactions

Emergency Stop

Receipt of the Emergency Stop instruction stops the locomotive as fast as possible. This sets the set speed to zero (preserving existing direction) and the commanded speed to zero (preserving existing direction) regardless of any momentum, BEMF or other operations with the train node.

Emergency stop is not specific state. The next Set Speed/Direction instruction will act immediately to change the set speed, and start the commanded speed and actual speed moving toward that set speed.

3 Background Information

3.1 Proxies

3.1.1 Lifecycle & Location

Proxies have a lifecycle:

1. They are created. This might be when a physical node (the proxy or a physical node hosting a number of proxies within it) comes up, or they might be produced as needed.
2. They get allocated to a particular legacy address
3. They are used to operate the equipment at that legacy address. (Although this is optional, it's the whole point of having the proxy)
4. They are deallocated. (Items 2 through 4 are a loop, and can happen multiple times)
5. Finally, they are destroyed. (This is optional; a proxy may stay around until the layout is powered off)

So long as the proxy can do it's job, there's nothing that requires the proxy to be resident in the command station hardware. It can be a separate board, in a computer somewhere that talks to pre-existing legacy equipment, or something else. One way for OpenLCB vendors to support DCC is to put the proxies and a command station in a single unit, but that's not required. Another approach would be an OpenLCB-compatible board that provides proxies, which then talk out through a LocoNet, XpressNet, TMCC or other connection to a legacy command station of a particular type.

The simplest way to provide N proxies is to have N nodes appear when the layout turns on. At any given instant, only one idle proxy is needed, so another approach would be to create them as needed. In this case, the newly created ones will get link access (e.g. go through the process of getting an alias on CAN), then announce their existence with an InitializationComplete message followed by ProducerIdentified and ConsumerIdentified messages as needed. Those can be used by Throttle nodes to track the NodeID of available proxies without having to query for one. Note that this process needs to be very fast, so that multiple throttles needing to allocate proxies at the same time can reliably do it.

3.1.2 Proxy Allocation

Mostly about location of an Idle proxy.

3.1.2.1 Collisions During Allocation

3.1.3 Consisting

Proxy can handle multiple node IDs (and/or DCC addresses), which allows implementation of consisting. Speed/direction just passes through; speed matching is automatic due to the use of scale meters/second for units.

It's up to the proxy how to handle pass-through of functions. (Do we need to have a way to specify which of the included nodes IDs (and/or DCC addresses) are special, e.g. front or rear locomotive, in the setup?)

It's up to the proxy how to handle pass-through of configuration read and/or writes. Probably just don't allow them, or allow them only for configuration of the consist itself (e.g. how to do certain pass throughs)

Table of Contents

