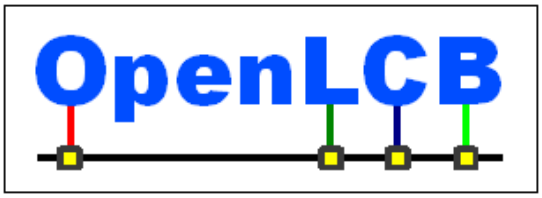


OpenLCB Working Note	
Traction Protocol	
Apr 18, 2014	Preliminary

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.

This protocol is for the long-term operation of really cool trains, not just "better control over DCC". Still, DCC is an important test case, both with new control hardware (OpenLCB native throttles connected to an OpenLCB-native command station that's generating the DCC signal) and when connected to legacy throttles and/or command stations. Using legacy hardware within the OpenLCB environment is accomplished through a proxy and is implemented using this Traction Protocol and the Traction Proxy Protocol.

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.

"Train Node": A Train is associated with a single, specific node. The Node ID is the fully-unique identifier for that Train.

"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, "proxies" 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. Another 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.

"Configuration", "Functions": Traditional DCC decoders provide "functions" for controlling accessories such as lights and sounds during operation, and provide a separate mechanism for doing long-term configuration via Configuration Variables (Cvs). OpenLCB makes the same distinction, providing access to "functions" via the traction protocol, while leaving "configuration" to the configuration protocol(s). The line between these is admittedly vague, and different node developers may implement some capability one way or the other. The general intent is that things that are changed in normal operation are considered functions, while things that are set once and forgotten are configuration.

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.

1.3 Unserved Use Cases

1.3.1 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

Node implementing the Train Protocol must implement:

- Message Transfer Protocol
- Event Transfer Protocol
- Memory Configuration Protocol (optional?) and Datagram Protocol it depends on
- CDI
- SNII and/or ACDI?

Float-16 is the half-precision numeric format defined by IEEE 754-2008. This is the format that the GNU tool chain's -mfp16-format=ieee flag and __fp16 type makes available on some CPU types.

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

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 0x05FA, 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.

Dedicated OpenLCB messages are defined for traction control, instead of using datagrams, so that they have higher priority. Given the small size of these messages, they also use less bandwidth than datagrams.

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	0x20							
Query Function	0x21	Address						
Controller Configuration	0x30	Assign Controller 0x01	Controller Node ID					
		Release Controller 0x02	Controller Node ID					
		Query Controller 0x03						
		Controller Changed Notify 0x04	New Active Controller Node ID					
Consist Configuration	0x40	Attach Node 0x01	Node ID(s) Multi-Frame?					
		Detach Node 0x02	Node ID(s) Multi-Frame?					
		Query Nodes 0x03	Node ID(s) Multi-Frame?					
Traction Management	0x50	Reserve 0x01						

Instruction	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
		Release 0x02						

The Set Function instruction uses a three-byte address for brevity; it's to be interpreted with a high byte of zero to make a four byte address in the function memory space (0xFA).

2.4.3 Traction Control Reply Message

MTI: Priority 1, index 15, modifier 0, addressed => MTI 0x05F8, 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	0x20	Set Speed		Status	Commanded Speed		Actual Speed		
Query Function Reply	0x21	Address			Value				
Controller Configuration Reply	0x30	Assign Controller Reply 0x01	Result: 0 == OK Non-zero == Failed	Fail Code: 1=Active Controller Refused					
		Query Controller Reply 0x03	Active Controller (0.0.0.0.0.0 if no controller active)						
		Controller Changed Notify Reply 0x04	Result: 0 == OK Non-zero == Reject						
Consist Configuration Reply	0x40	Attach Node Reply 0x01	Node ID (TBD Should this be a Datagram to send multiple nodes at once?)						Reply Code
		Detach Node Reply 0x02	Node ID (TBD Should this be a Datagram to send multiple nodes at once?)						Reply Code
		Query Node Reply 0x03	TBD What to do here... Should this be a datagram??						
Traction Management	0x50	Reserve Reply	Result: 0 == OK						

Instruction	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Reply		0x01	Non-zero == Failed						

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

There's no reply defined to Set Speed and Set Function to reduce bandwidth use. OpenLCB is a reliable-transport network, so these replies are not required for reliability. Train nodes must be able to receive and process them at full rate, as they can arrive adjacent to each other on the link.

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

which together are the single message

MTI=05E8 10 44 20 00 44 20 44 20

Controller configuration maintains the node that is controlling the train node. In order to operate a train the controller must be first assigned. A train node shall not respond to a traction message if it not defined as the current active Controller. Once a controller is finished operating a train it should, but is not required, to Release the active Controller. One place this may not be done is if the controller is being unplugged to be moved to a new location on the layout. The benefit of releasing the train is to allow the next controller to be connected to the train without the handshaking required to take a train from an existing active Controller. This is explained in the messaging diagram later in this document. A Controller node may be send an unsolicited message if it is currently assigned to a train node and another Controller node attempts to assign itself to the train. The when new the Controller node requests to be Assigned to the train one of several actions may occur.

1) The train may block the request and return false to the Assign request

2) The train forwards a Controller Changed Notify message to the currently assigned controller node and that controller node returns false in which the train node returns false with a fail code of Controller Refused to the Assign request. The active controller may, but is not required to, indicate to the user it has been asked to release the train node.

3) The train forwards a Controller Changed Notify message to the currently assigned controller node and that controller node return true. The active controller may, but is not required to, indicate to the user it is loosing its position as the active controller of the train node. In turn the train node returns true to the Assign request and the train node sets the new controller as the active controller.

A node my Query the active node from a train node. If no controller is assigned the train nodes returns a Node ID of 0.0.0.0.0.0. It would be better to return an error code but that would require a multi-frame message which is possible but undesirable.

Consist Configuration allows a train node to become the operator of multiple trains. The simplest way is to Add/Delete/Query one node at a time. I think this should be done as a datagram for all operations. It is not a time critical operation and the number of nodes should not be artificially limited to force it into a multi-frame message.

Traction Management locks/releases the train node such that interactions that require multiple messages between 2 or more nodes are guaranteed to be completed before another node can set (modify) the configuration on the train node. These messages are not needed for normal traction operations such as speed and function set/query. If the node attempting to set attributes of the train is not the active node the train node will ignore the request and as such locking/releasing is not required.

2.5 States

Full OpenLCB nodes:

- Set Speed – The speed set by a throttle, the content of the most recent “set speed” instruction
- Commanded Speed – the current speed that is intended. This may lag behind the set speed intentionally due to momentum settings, etc.
- Current Speed – a physical state, the speed at which the object is currently moving

Emergency stop is not a state.

2.6 Memory Spaces

2.6.1 Configuration Information

The configuration memory space holds the configuration of the train, such as how functions will work, how speed in scale meters/second will control motor operation, etc.

2.6.2 CDI

2.6.3 Function Information 0xF9

Functions, such as lights and sounds, can be operated by the Traction Control Set Function instruction, and their current value can be retrieved via the Traction Control Query Function instruction. The values are also available for reading and writing in the Function Information memory space. This allows

multi-byte reads and writes using Configuration Memory Protocol access. That's a more efficient way of setting and reading large numbers of functions, for backup, initial setup, etc.

The NMRA 9.2.1 Recommended Practice describes DCC as having three separate sets of "functions". The most common one is the traditional F0-F28. In addition, the "Binary State Control Instruction long form" accesses 32767 addresses (confusingly called "states" in the NMRA doc) and the "Binary State Control Instruction short form" accessed 127 addresses (the NMRA document implies that these are overlapping address spaces, but at least one manufacturer has not implemented them that way). Finally, NMRA DCC WG Topic 9910241 never made it to the Recommended Practices due to internal Working Group politics, but has been widely implemented by decoder and command station manufacturers. It provides an "Analog Output Instruction" or "Analog Function Group" with 8 bits of address space and 8 bits of value for each address, for a total of 256 functions and a value in the range [0..255] for each function.

The OpenLCB 0xF9 space is allocated to cover all these by using the third byte of the address as a selector.

Type	Low Address	High Address	Values
F0-F28	0x0 00 00	0x0 00 1C	A non-zero value indicates "ON", a zero value is "OFF".
Binary State Controls (full space)	0x1 00 00	0x1 7F FF	"
Binary State Controls (short space, if separate)	0x2 00 00	0x2 00 7F	"
Analog Outputs	0x3 00 00	0x3 00 FF	

The default Function Definition Information, when nothing is known about a particular decoder's capabilities, will just describe these as above. If the node serving the DCC decoder information knows more about the particular decoder's capabilities and/or preferred labeling, it can provide more information via the Function Definition Information.

There's nothing in the standard that says that the first three types of information need to be stored in 29+32768+128 bytes. Packing them into bits is certainly acceptable, given that the underlying DCC protocol can only send one bit for each. An individual node may not implement all of them, either.

2.6.4 Function Definition Information FDI 0xFA

“Function Definition Information”, similar in intent to Configuration Definition Information (CDI) is stored in XML format in address space 0xFA to provide user-oriented context. That includes:

- Memory layout of the function values, allowing for multiple data types from binary (one and off for lights) through integer values (for e.g. sound intensities) and strings (sign displays?).
- Function naming, so that a throttle can display useful names to the user such as "Bell", "Coupler Clank" and "Master Volume". This includes internationalization of those labels.

What else needs to be conveyed? "Make this prominent on the throttle"? "Have this there, just a little less prominent"? "Seriously, nobody cares about this option, bury it"?

At present, there are no default values that e.g. associate "Bell" with a particular location or function. These are thought to be too brittle, and there are just too many possibilities to be useful (see the unscientific and incomplete Survey of existing function names).

2.7 Interactions

2.7.1 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.

2.7.2 Function Operation

Function values are stored in the 0xF9 memory space. They are written using the memory configuration protocol or using the Set Function instruction.

2.7.3 Train Configuration

Trains are OpenLCB nodes just like any other. As such, the Memory Configuration protocol can be used to configure them, the Configuration Description Information system can be used to make that process user friendly, etc. There's nothing traction-specific in these techniques, which are available any time that the train node is connected to the OpenLCB.

The configuration information in a train can include the user documentation that's sometimes referred to as "roster information". This might include owner name, prototype railroad and road number, information about the particular model's construction, etc. As yet, OpenLCB has no standards nor conventions on this information.

One approach to standards in this area would be extend ACDI/SNIP. Those currently have a block for manufacturer identification, and a block for user identification. Those are both versioned. We could take (one of) several approaches:

- Extend them with a third block, only present when the node implements the train protocol (as seen in PIP). To allow later introduction of more types, this block would have some versioning/type information, but that's straightforward.
- Create a version 2 of the user block, which holds additional data. (This would be using 2 as a format identifier, rather than a version number; that might make versioning complicated)

Suitable content for a (first version of) this might be (from [http://jmri.org/JavaDoc/doc/jmri/jmrit/roster/RosterEntry.html JMRI roster], see also similar concepts in [rocrail.net RocRail]):

- Road Name
- Road Number

(Manufacturer, model, owner description, comments, etc are already present in ACDI/SNI)

For DCC locomotives, more terms might be desired:

- DCC Address
- Decoder Type (Manufacturer, model)

2.7.4 Train Identification

OpenLCB Train nodes use:

- The Event Transport protocol to locate Train nodes
- PIP for enquiry about the support
- SNII and/or Memory configuration, CDI & ACDI for identification of a specific train node.

Trains are OpenLCB nodes just like any other. As such, they can take part in protocols such as Node Verification and Simple Node Information which allows other nodes to learn about them.

Train Acquisition Protocol is necessary because the train operator doesn't want to pick up a throttle and enter “06.011.00.02.1F.2D” (a node ID), or even “110 Long” (a DCC address), but rather just pick the desired locomotive from a list of those available. (A throttle should still allow the operator to directly enter the address, when that's what the operator wants to do.)

The train acquisition process simplifies locating desired train nodes so that small hand-held throttle nodes can efficiently take part. It does this using several approaches, which can be used as needed by throttles:

- Events are used to announce the existence and status of Train nodes
- Train nodes will generally implement the Simple Node Information protocol so that throttles can get basic, user-readable identification from them
- A search protocol is (being) defined to make it possible to locate individual Train nodes without having to read information from all of them

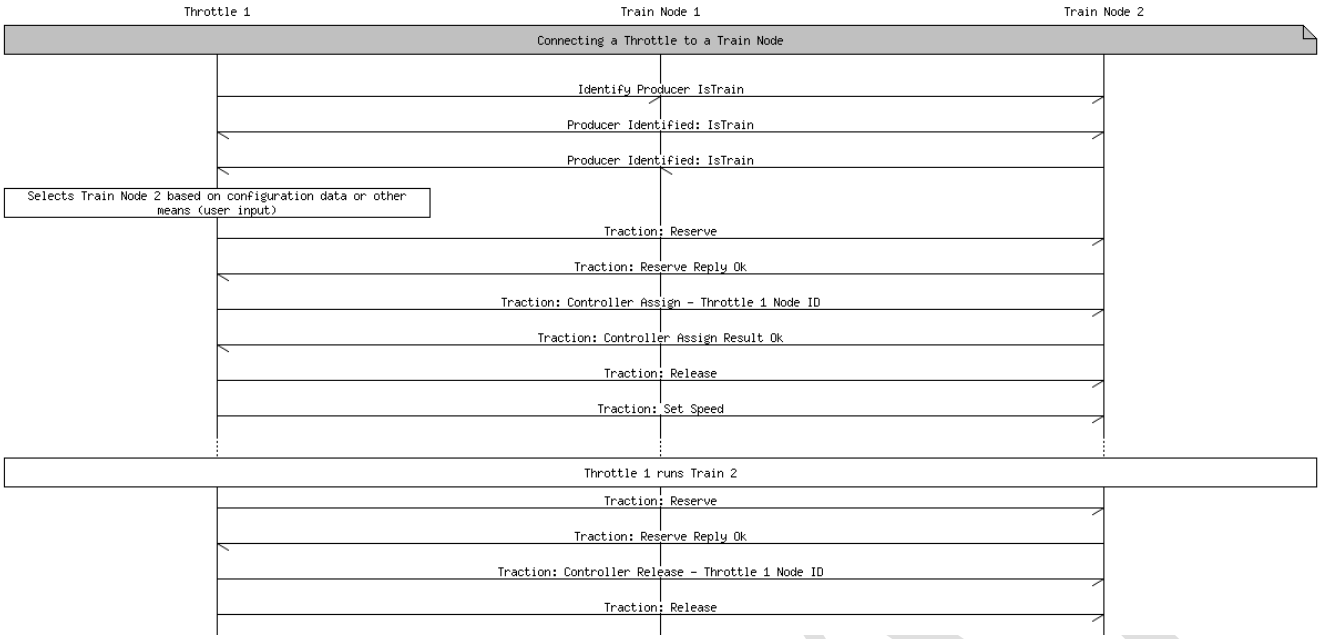
So that other nodes can find them, Train nodes must produce the well-known reserved event 01.01.00.00.00.00.03.03. This means they must produce that event in a Producer/Consumer Event Report message when they power up, and reply to requests for producers of that event. An IdentifyProducer request will therefore find all the train nodes on the OpenLCB, and further protocols can be used to get additional information on the individual Train nodes it locates.

SNIP will be used to carry both manufacturer-provided and user-provided information about the particular train node. In particular, the user (Node) Name and (Node) Description fields are to be used to hold train identification information that can be retrieved and presented by throttles for selection.

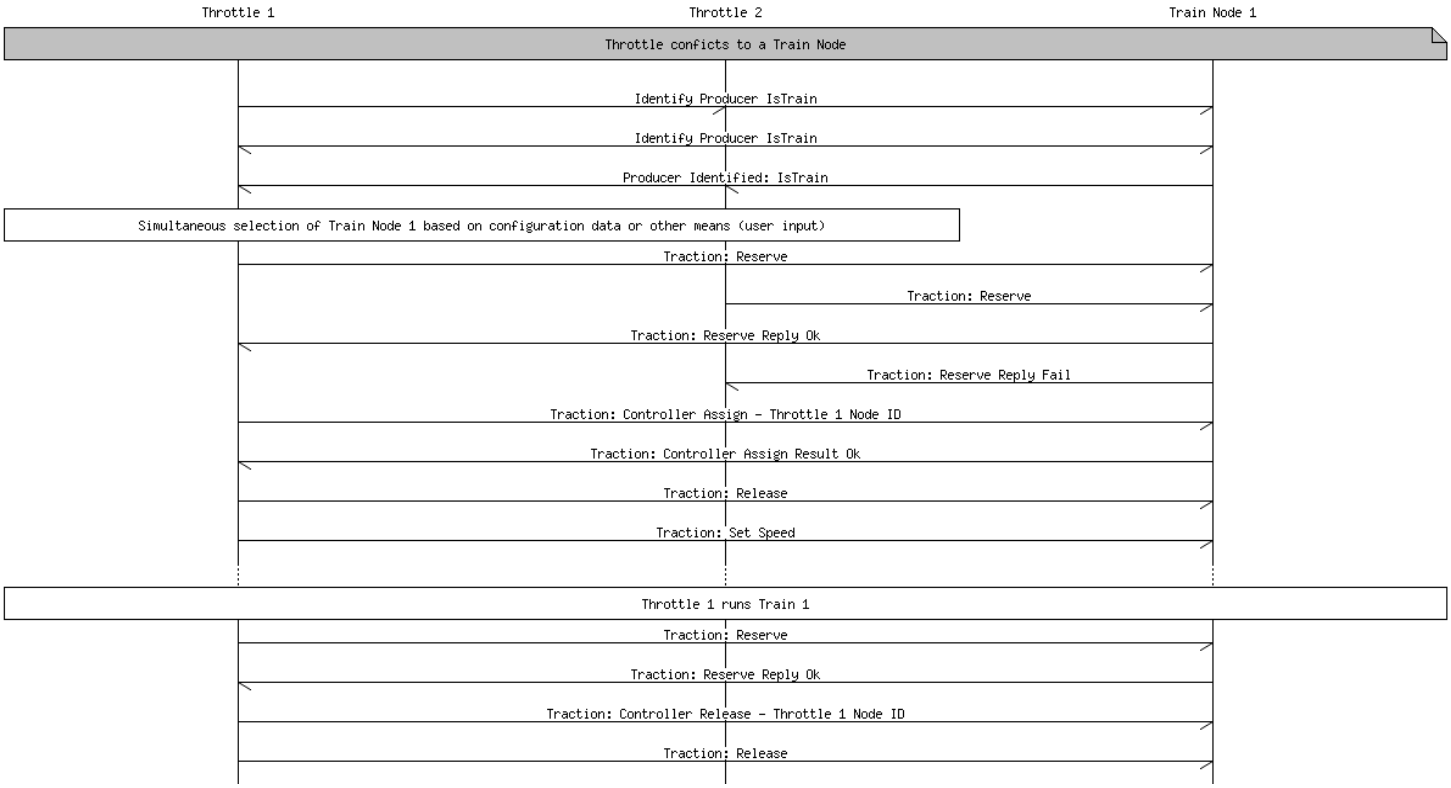
A search protocol maay eventually be defined to allow efficient location of specific train nodes on large OpenLCB installations. It provides a general or field-specific search over the SNIP information,

returning the Node IDs of matching Train Nodes. One proposal, which is not based on memory configuration & SNIP, is available here in pdf format and here in OpenOffice Writ

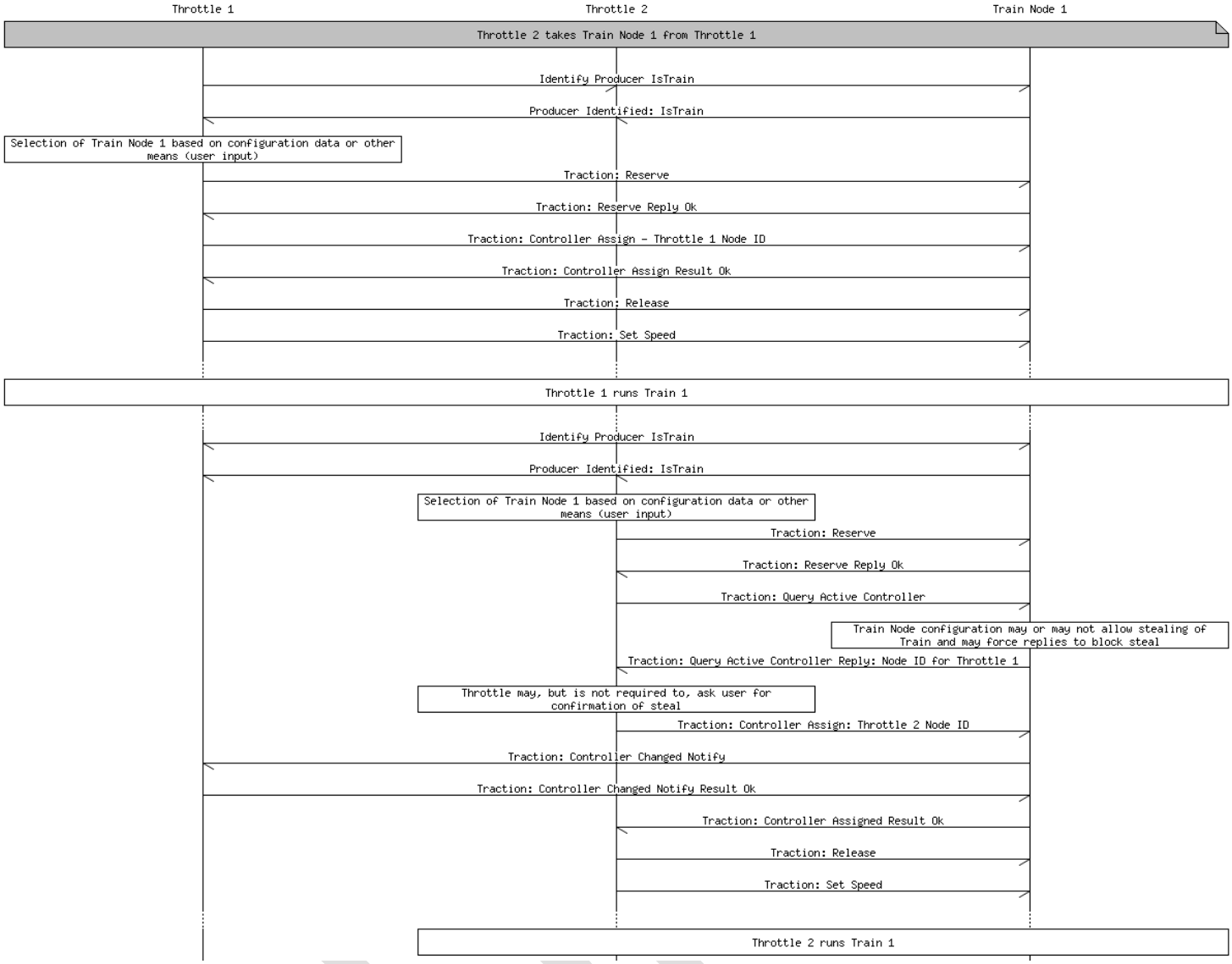
2.7.5 Basic Throttle and Train Connection



2.7.6 Conflicting Throttles



2.7.7 Throttle to Throttle Successful Hand-over (steal)



3 Background Information

3.1 Speed Control

For OpenLCB, the speed and direction to set is encoded as a half-precision floating point number (aka 'float16'), with positive numbers indicating forward direction, negative indicating reverse, and (signed) zero indicating full stop. The value specifies a speed in scale meters per second (scale-m/s).

Note that even the zero value is signed. This is needed because locomotives still have a direction, even when they are fully stopped. It's used to control the configuration of lights and sounds that the locomotive exhibits when stopped. “Negative zero” is well-defined in the IEEE float-16 standard, but

not all libraries implement it well, and it's easy for code to convert the negative-zero value to the more common positive-zero by default. Setting the sign after all computations are done is one way to handle this.

Rationale: The use of a 16-bit floating point permits relatively precise speed commands, especially at lower speeds; such fine granularity ensures not just fine-grained control over the locomotive, but helps avoid aliasing issue that arise during the conversion to lower resolution system-specific speed commands (i.e. DCC's 14 or 28-step commands). Using 32-bit floats uses more bandwidth, more program and data memory and CPU cycles on small nodes, but is somewhat easier for large nodes. The conversion between float-16 and float-32 is very simple, though, and any node with native 32-bit support can handle float-16 easily. The converse is not true.

The use of meters per second is somewhat arbitrary, and reflects standard velocity units used throughout the metric-speaking world. By standardizing on specific units, we avoid any future unit conversion issues. By standardizing on metric units, we simplify future attempts to simulate and control train physics.

The use of *scale* meters per second has two distinct advantages. First, it permits us to transmit speed commands in a scale-independent way. Second, and because of this, it reduces the number of parameters that must be estimated when controlling a locomotive that has not yet been speed-calibrated (which, for new users using existing digital control systems, will be all of their models). For example, on a DCC system, if I issue a command to proceed at 30mph, the command station must convert the value in the speed command from 30mph to an integer in the range [0-26] (for 28-speed-step control). The command station need only estimate what a reasonable top speed for a locomotive might be: Let us say, 100mph. Thus, the command station could reasonably estimate that 30mph translates to speed step 8.

The alternative possibilities considered were absolute speed using real units (as opposed to scale units), and relative speed units. The difficulty with relative speed units (i.e., percentage of full throttle), is that they are ambiguous, and preclude the possibility of performing physical simulations in the cab controller, at least without completely abandoning the particular interpretations assigned to sped values. The difficulty with using real (as opposed to scale) units is that it requires the estimation of an additional parameter for uncalibrated locomotives, specifically the train's scale. If I issue a command to a DCC locomotive to proceed at 0.1 (real)m/s, the command station must not only understand what a reasonable top speed for a train is, but how to scale the speed appropriately, as 0.1 m/s might be quite fast for Z scale, but quite slow for G. As there is really no reasonable scale to use as a default, users must configure their digital command station to set the scale for either the entire layout, or on a per-model basis—an additional configuration step that is easily avoided by the mechanism for scale units described above.

3.2 Function Control

"Functions" like "horn", "headlight", etc are key user features when operating modern decoders. But they're also configuration-like, in that they effect the operation of the device.

How to handle them for a native OpenLCB piece of rolling stock, and for legacy ones via e.g. DCC?

We could take the purist approach and say "configuration is configuration, it's all the same". But that ignores that many people are going to want "Bell" to appear (automatically) on their throttle, but not so many are going to want "Kp back-emf correction factor" to appear there.

People just think about operating and configuring their locomotives as separate things. (Though e.g. "Master Volume" can cross the line)

This doesn't mean that we can't use the same protocol for all of them. A mixture of memory configuration and CDI-like definition information should do just fine. It just means that we need to find a way to include clueing information for the throttles on e.g. what to present.

Many trains offer independent control of various special effects (FX, sometimes called "functions") such as lighting or sound. The Set Function instruction permits a controller to control these effects directly. The first argument is the address of the FX to control as a 24-bit unsigned integer. This protocol does not define a semantics for FX addresses; that is, there is no particular address that is singled out as representing headlights or the air horn. Instead, the addresses are deliberately abstract, permitting the user to decide how to map addresses to FX for each train.

Additionally, each FX can take a 16-bit value. Current technology only permits binary FX; thus 0x00 should be interpreted as "off" for a binary FX, and any non-zero value as "on". Analog (non-binary) FX should treat the 16-bit value as an unsigned integer.

Function values are stored in the 0xF9 memory space. "Function Definition Information", similar in intent to Configuration Definition Information (CDI) is stored in XML format in address space 0xFA to provide user-oriented context. That includes:

- Memory layout of the function values, allowing for multiple data types from binary (one and off for lights) through integer values (for e.g. sound intensities) and strings (sign displays?).
- Function naming, so that a throttle can display useful names to the user such as "Bell", "Coupler Clank" and "Master Volume". This includes internationalization of those labels.

What else needs to be conveyed? "Make this prominent on the throttle"? "Have this there, just a little less prominent"? "Seriously, nobody cares about this option, bury it"?

At present, there are no default values that e.g. associate "Bell" with a particular location or function. These are thought to be too brittle, and there are just too many possibilities to be useful (see the unscientific and incomplete Survey of existing function names).

3.2.1 Outputs vs Functions

Tools like DecoderPro and its decoder-definition files make a distinction between "functions", which are the control commands sent via e.g. DCC, and "outputs", which are the things that a decoder can do: Control an electrical output, make a sound, etc. This distinction is useful because one of the configuration options in (some) DCC decoders is a mapping between the functions and the outputs, useful in a world where throttles generally have only about a dozen buttons, but decoders have many output options.

OpenLCB makes a clean separation between functions, which are the control operations, and all configuration & physical information, which lives in the memory configuration and CDI. If there's to be a mapping, it's defined through the CDI.

3.2.2 Legacy Function Control

Although currently most DCC throttles only permit access to 12 or 29 functions, DCC technically permits as many as 32796 different binary FX (see RP-9.2.1: Function Group One Instruction, Function Group Two Instruction, and Feature Expansion Command Instruction especially the Binary State Control Subcommand). For this reason, it seems prudent to go ahead and use a 24-bit value.

Likewise, although current DCC decoders only permit binary FX, some (for example SoundTraxx Tsunami sound decoders) actually permit a kind of analog control of FX by combining multiple DCC Functions. Thus, it seems likewise prudent to permit a wide range of values, and not simply a binary on and off.

One problem that faces the decision to use a single command with a numerical FX addressing system is that any kind of standardized assignment of FX addresses to particular FX is impossible in practice. DCC, for example, makes no such prescription, although by convention function F0 controls direction-sensitive headlights. Beyond this lies only manufacturers' conventions. Thus, any kind of mapping is best handled on a per-train basis, by configuring the mapping between particular FX (e.g. headlights, air horn) to FX addresses for each train.

One way to mitigate this problem is to not make fixed FX address assignments, but to map them directly onto the addressing scheme used by the various control systems, that is, to leave each address in the OpenLCB FX address space uninterpreted. In this way, the default behavior of each address will map directly onto the default behavior of the decoder in the train, giving some degree of predictability to the system, and permitting a digital command station the make reasonable guesses about the possible address-to-FX mappings. Nevertheless, users will often need to be exposed to this implementation detail, which is deeply unfortunate, but necessary given the ultimate flexibility of current train control systems.

Thus, it is recommended that the FX address space be mapped directly to the particular control system's address space; thus DCC F0 becomes FX address 0x000000, F1 becomes 0x000001, etc. The DCC Binary State addresses should be mapped to 0x010000 to 0x017FFF (i.e., to the 15-bit range beginning with 0b1.0000.0000.0000.0000). Other systems should be handled analogously.

RP-9.2.1 defines the following:

- Function Group One: 5 (F0-F4)
- Function Group Two: 8 (F5-F12)
- Binary State Control: 32767 (15bits) (note: different address space!)
- Feature Expansion Command 11110: 8 (F13-F20)
- Feature Expansion Command 11111: 8 (F21-F28)

And in the future, perhaps even more. Possibly a lot more. There are just under 20 bits of address space available in the Feature Expansion Command Instruction for the potential manipulation of Binary State Controls.

3.3 Configuration

Trains are OpenLCB nodes just like any other. As such, the Memory Configuration protocol can be used to configure them, the Configuration Description Information system can be used to make that process user friendly, etc. There's nothing traction-specific in these techniques, which are available any time that the train node is connected to the OpenLCB.

The configuration information in a train can include the user documentation that's sometimes referred to as "roster information". This might include owner name, prototype railroad and road number, information about the particular model's construction, etc. As yet, OpenLCB has no standards nor conventions on this information.

One approach to standards in this area would be extend ACDI/SNIP. Those currently have a block for manufacturer identification, and a block for user identification. Those are both versioned. We could take (one of) several approaches:

Extend them with a third block, only present when the node implements the train protocol (as seen in PIP). To allow later introduction of more types, this block would have some versioning/type information, but that's straightforward.

Create a version 2 of the user block, which holds additional data. (This would be using 2 as a format identifier, rather than a version number; that might make versioning complicated)

Suitable content for a (first version of) this might be (from [http://jmri.org/JavaDoc/doc/jmri/jmrit/roster/RosterEntry.html JMRI roster], see also similar concepts in [rocrail.net RocRail]):

- Road Name
- Road Number
- (Manufacturer, model, owner description, comments, etc are already present in ACDI/SNI)
- For DCC locomotives, more terms might be desired:
- DCC Address
- Decoder Type (Manufacturer, model)

3.4 How to find human-readable loco information

e.g. SNII for getting the name of a real loco, DCC loco. Examples of how to provide a human-readable roster on the throttle. Interactions with configuration.

I believe we should strive to make our proposed system not be specific to a particular digital system (e.g. DCC) or otherwise biased. The procedure of "take the cab number from the loco, punch it in and drive" is actually quite US-specific.

Balazs Racz wrote about how identification works in Europe:

Each piece of rolling stock is labeled with a unique UIC number. UIC number is a 12 digit number which identifies the type of vehicle, the country in which it is registered, the owner, the type of vehicle and a unique number in its series. It might look like

91 85 0 474 014 2

This number is written in a size on the prototype that is legible to a person standing next to it. It is very much not obvious to read it off of a model. this number is not really suitable to punch into a throttle in order to get going...

[Note about assumptions: Freight cars also have such an ID. However, JMRI itself limits the length of the car ID t something like 8 or 10 characters, so it is not possible to accurately represent a European freight car in JMRI.]

If you take the shortcut of removing the type and country code, then you end up with 474014, which is a common way to refer to a locomotive say within Switzerland. However, this is too long for a DCC address! So I don't have the choice of putting in the cab number as address. (Of course now your numbers are not unique, for example see http://cfr.stfp.net/Pic/47/915304740144DQOM:1.jpg and http://www.bahnbilder.de/1024/sbb-fahrzeugausstellung-bei-sbb-459408.jpg)

Another example: MFX is the current track protocol of Marklin. MFX is bidirectional, and locomotives sign in with the command station as soon as they get put on the track. MFX uses 11, 14 or 28 bit addresses. The locomotive address is assigned by the command station upon the login of the locomotive. It is not persistent. The user never sees the locomotive address and in fact it may even be changed by the command station if it so desires.

So punching in the loco address into the throttle is also not an option.

Example 3: Older Marklin protocol (Marklin-Motorola protocol or MM) supported only 79 different addresses on the track. Then the address on the track had again nothing to do with the actual engine number, and you don't need to have an extraordinarily large collection to run into duplicate addresses. However, owners of such collections do typically remember the two-digit address of each locomotive they have, because older command stations only had digit buttons for selecting a locomotive.

I would find it a shame if the recommended design of OpenLCB would only support US-based roads.

State of the art command stations have color touchscreens where they display you a list of locomotives that you can select from with a fingerpress and get running. You either populate this roster by hand, or it gets populated automatically for locomotives that log in via track feedback systems (like MFX or RailcomPlus). Typically pictures are also assigned to each locomotive.

Note that SNII/ACDI defines a manufacturer as < 41 bytes, model name < 41 bytes, hardware & software versions < 21, node name < 63 bytes, node description < 64 bytes (including terminating null), much more of than Balazs' example.

3.5 Consisting

Consisting is a process familiar to model railroaders, who use it for many purposes:

- Run multiple diesel engines together as a model MU
- Run helper locomotives together due to a shortage of (real) engineers
- Connect multiple pieces of rolling stock (passenger cars, cabooses) together to make it easier to control sound and lighting effects
- Connect sound decoders with motion-control decoders so they work well together

OpenLCB handles consisting and similar “one acts for many” situations by having a node act as the front-man for the consisted group. This node may be a real physical object, but it's more likely to be a software construct somewhere within the hardware that's used to connect the main OpenLCB network to the actual trains.

Once the consisting proxy node has been initialized, it will handle the individual parts of the Traction Protocol series:

- Speed Control - this is perhaps the simplest, just passing the speed values through to the individual members of the consist. Because OpenLCB speeds are in scale meters/second, there's by definition no need to rescale them when forwarding them to disparate equipment
- Function Control - Although the setup process may be able to map or reassign functions in the consist-member nodes, in the end this protocol is just a pass-through of the function instructions and memory access protocol operations.
- Train Configuration - except for very limited configuration of the consist itself, the consist does not take part in any configuration operations. Those are done by talking directly to the nodes that control the individual pieces of the consist. (CDI for the consist node is going to require careful definition)
- Train Acquisition Protocol - the consist is a separately locatable thing, as throttles will want to be able to find it. As such, it will take part in producing the well-known events, providing human readable (and writable) information via SNII/ACDI, etc, and being the target of search operations.

Because there's no protocol difference between full-OpenLCB-node Trains and legacy equipment that uses OpenLCB proxy nodes, consisting works the same for all of those. It can even mix-and-match full nodes with various types of legacy trains, if that's electrically possible. Proxies 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.

Table of Contents

