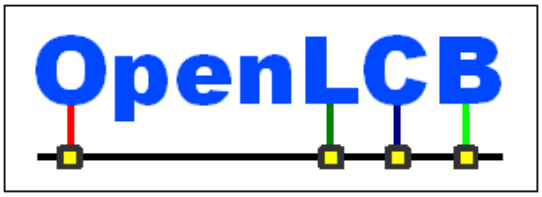

OpenLCB Technical Note	
Traction Protocol	
Sep 2, 2022	Preliminary

1 Introduction

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

1.1 Served Use Cases

1.1.1 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.

1.1.2 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.

1.1.3 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 Unserved Use Cases

1.2.1 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 Annotations to the Standard

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

2.3.1 Terminology

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

2.4.2 Traction Control Command Message

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.

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.

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.

2.4.3 Traction Control Reply Message

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

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

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 is a bit set. Bit zero shall be set if and only if an Emergency Stop command was received after the last Set Speed/Direction command. 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

2.5 States

2.6 Interactions

2.6.1 Train Assignment

A node may Query the active node from a train node. It would be better to return an error code but that would require a multi-frame message which is possible but undesirable.

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.

2.6.2 Emergency Stop

2.6.3 Function Operation

2.6.4 Train Configuration

2.6.5 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

2.6.6 Basic Throttle and Train Connection
2.6.7 Conflicting Throttles
2.6.8 Throttle to Throttle Successful Hand-over (steal)
2.6.9 Listeners

Listener Configuration allows adding listeners to a train node, where the train node shall forward the state-changing requests to. Listeners always get speed change commands forwarded (direct or with reversed direction), and optionally Function 0 or all function commands as well.

A train node may provide additional configuration, for example as part of the CDI to customize the forwarding of messages. This may be interesting for virtual train nodes specifically designed to be consist proxies.

2.6.10 Heartbeat

Train nodes may employ a heartbeat mechanism to ensure that the assigned Controller (typically a throttle) is alive and in control of the train. This is a safety feature that protects valuable equipment from technical failures causing a runaway train rolling uncontrollably.

The Heartbeat Request shall be initiated by the train node. This choice was made in order to allow centralizing the timeout based logic in one participant. It is recommended to make heartbeats a selectable option for a train or proxy (command station). The train shall not initiate a heartbeat request if it has received a command or query from the Controller node within the time period of the heartbeats.

Alerter mode. The Standard purposefully does not specify that a Controller shall automatically clear the heartbeat request. It is a valid implementation to require the operator of the train to clear the heartbeat and provide an appropriate user interface to do so (e.g. audible or visual alert and acknowledgment button). This is prototypical behavior for real locomotives, also called “dead-man switch”.

There are two timing parameters in the Heartbeat interaction, which may be specified separately by the manufacturer of the train node or the configuration of the user.

- The time period of heartbeats: the maximum idle period after which a heartbeat is sent.
- The deadline: after a heartbeat is sent, how much time does the Controller have to reply.

The suggested default values are 10 seconds for period and 3 seconds for deadline. It is recommended for manufacturers to make the period configurable by the user.

2.7 Memory Spaces

2.7.1 Configuration Information
2.7.2 CDI
2.7.3 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.

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.

2.7.4 Function Definition Information FDI 0xFA

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 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 DCC 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)

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. The Simple Train Node Information Protocol (STNIP) has been defined to implement this concept.

3.3.1 Legacy DCC Configuration and CVs

[Moved here from the old Wiki Page]

OpenLCB has a protocol for handling configuration of nodes. It should work fine for native OpenLCB rolling stock.

Can/should it be adapted to handle the way existing DCC locomotives are configured through CVs?

For CVs, one idea that we'd have a memory space that maps straight to the decoder CV space.

That would allow decoder-specific CDI (if that was available) to customize what's presented. Like DecoderPro, the user wouldn't have to deal with CV 111, but rather with "Motor multisnarb angle offset" or whatever the user-fiendish manufacturer decides to provide as options.

There are two remaining issues. First, is handling "indexed" CVs. (Ones where you write 12 to CV 51, then 8 to CV52, and then CV54 is the value you want to read/write). QSI is the only extensive user of these, but they're starting to appear in other places too.

There are a couple ways to handle it. First, we could just map it as is: The configuring node would have to explicitly do those reads and writes. But that's a mess, well outside the OpenLCB model, and I've spent way to much time debugging weird failures with that.

A better approach, I [Bob J] think, is to use the large address space. E.g. CV 59 is found at 0x00 00 00 3B, while the one I mentioned above is found at 0x01 00 0C 08. (The 0x01 tells how to decode the address space, e.g. which CV the 2nd byte is written to, which the 3rd byte is written to. We'd have to extend CDI to carry that info, but it's within reason.

The other issue is packed CVs, e.g. mapping parts of one or more CVs to a single "variable". This could be just a single bit in one CV, or something more complicated split across two (like long addresses). I'm not sure how to map those as a general case. For bits, I'd suggest another mapping trick, where some other part of the space is actually bit mapped:

0xFF 00 53 3C

is the middle bits (the 3C mask) of CV 53, and the 0xFF is just an arbitrary key for this. That doesn't work with noncontiguous splits across generic CVs, though, as that takes a lot to configure.

(And don't get me started on CV1/CV29 sequencing; I think we just ignore that entirely at the CDI level & build it into the gateway to DCC)

In the end, these legacy pains can't really be avoided. We have to craft the OpenLCB protocols to do a reasonable job with them, but I don't think we have to go nearly as far into the weird special cases as e.g. DecoderPro does.

3.4 How to find human-readable loco information

A search protocol may 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 Writer

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 and Listeners

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 by introducing the “Listener” concept to all train nodes. A listener is attached to a given train node in order to receive a copy of all state changing requests that arrive at the given train node. This serves multiple purposes:

- Consist follower engines can be attached directly to the consist lead train node and receive all speed set commands, which will cause the consist to move together.

- Throttles that lose the control of a locomotive (through stealing or hand-over) can register themselves as listener on the locomotive, and update the UI as the other engineer is driving the train.
- It becomes possible to add a display to the train node that displays a train's status and keeps it up to date without polling for updates and unnecessarily consuming bus bandwidth.

Consisting can be applied by two methods:

1. If the lead engine supports listeners, then the consist followers can be added directly to the lead as a listener. Note that the forwarded message is defined in a way that the consist follower sees it as if it came from a regular throttle, therefore the consist follower engine is allowed to not be aware of the consisting.
2. If no engine in the consist can be chosen as lead engine, then a virtual node can be instantiated that supports listeners and has the ability to attach all consist members. Such virtual nodes may be provided by throttles or command stations.

The listener configuration allows selecting some common use-cases for the consist:

- Forward speed only.
- Forward speed but invert direction (when the following locomotive is backwards-facing in the consist).
- Forward function zero (usually light). Turning this on for the lead and the tail engine in the consist, but off for all middle members of the consist will cause the consist to behave prototypically with headlights.
- Forward all functions. Useful for the throttle listener use-case or in case the two locomotives are equipped with identical decoders and features.
- More advanced configuration (e.g. function remapping, arbitrary function filter) may be applied by using the regular configuration user interface of the consist lead node (e.g. CDI). Such advanced configuration may be however cumbersome for the user to access, since dumb throttles may not have the ability to render arbitrary CDIs, while specialized user interface could be developed according to the above listed common options. Also creating and deleting consists may be considered as an everyday operation to users, therefore advanced consist settings may be accidentally destroyed. An appropriate save/restore feature may be necessary.

The consist's listener relation may be established in a symmetric manner, where the lead engine is also attached as a listener to each individual follower engine. Symmetric listener relations are allowed and cause no problem for the network, because a train node is required to never echo back a packet to where it came from.

The benefit of a symmetric listener relation is that now the consist can be driven by calling up any train number in it, and the control packets will be propagated to all members of the consist. This may be helpful if the user wants to drive a consist by the tail engine (for example after a runaround move) and use e.g. the horn feature on the tail engine. Of course this feature will only work if the tail locomotive supports listeners, otherwise it is impossible to establish a symmetric listener relation.

It is important that the symmetric listener relations create a spanning tree in the consist nodes. If there is any directed circle (longer than 2 nodes), any message injected will be forwarded between those nodes forever.

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 relationship 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 a specialized consist node is going to require careful definition)
- Train Acquisition Protocol - the consist, if controlled by a special node, 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. Consists driven by their lead engine can be operated by calling up the lead engine in a throttle. Consists with symmetric listener relationships can be driven by calling up any single node in the consist. Defining the assigned controllers may be tricky in such a situation.

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.

475

Table of Contents