OpenLCB Technical Note	
Train Control~~Traction~~ Protocol	
June 22, 2024	Preliminary

1 Introduction

This Technical Note covers the ~~Traction~~Train Control Protocol, the way that OpenLCB handles moving objects such as locomotives, engines, and other rolling stock. This document is intended to be read in parallel to the matching Standard, as commentary and background information.

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.

1.1.4 DCC Layout

Elena is running her DCC layout with an OpenLCB Command Station. Elena picks up an OpenLCB throttle, presses the locomotive button and sees the list of DCC locomotives currently running on the layout, with the addresses augmented by the respective locomotive name from the Command Station memory. Elena selects “C&NW 415” from the list and starts running her train. Elena forgot which function number turns on the Compressor, so she presses the Help button on the throttle. The throttle displays all functions of the locomotive with their description, so she knows she has to press 9.

DRAFT

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

1.1.5 Supervised running

David has his little nephew Noah visiting, who wants to run a train. David gives an OpenLCB throttle to him, which already has the locomotive dialed up. David takes a second throttle, and selects the same locomotive. David can see on his throttle real-time as Noah controls the train, and gently reduces the speed as needed. When they are about to derail on a switch set against them, David presses the emergency stop button on his throttle to prevent the accident.

1.2 Unserved Use Cases

1.2.1 DCC locomotive allocation and deletion

A DCC Command Station represents the current set of active locomotives on the OpenLCB bus using Virtual Train Nodes. These can be selected by an OpenLCB throttle.

There is no provision in this standard on how to make the Command Station create a new Virtual Train Node from a new locomotive address. This is covered in the Train Search Protocol.

There is also no command in this standard to instruct a DCC Command Station to delete a Virtual Train Node.

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

Additional commentary:

- Train Node:
  - The Node ID of the Train Node is the fully-unique identifier for that Train on the OpenLCB bus.
  - The simplest form of a Train Node is when a model has a decoder built into it which directly connects to the OpenLCB bus, presumably using wireless communication.
  - An important class of Train Nodes is when the Train Node is not built into the model, but instead remotely actuates the model using electrical signals or a legacy communication protocol. In this case typically one physical hardware connected to the OpenLCB bus represents many different Trains, each with its own unique **Virtual Train Node**. This is a form of a Gateway, bridging OpenLCB to a legacy protocol. From the perspective of the legacy protocol, this piece of hardware is often referred to as a **Command Station**.
- Throttle Node:

DRAFT

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- 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 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.
- A single Throttle Node may control multiple trains simultaneously and independently, for example when it has appropriate operator interface (e.g. multiple knobs, or multiple windows open on a computer screen).
- A single Throttle Node may also be a Gateway from a legacy throttle bus, translating commands originating from multiple physical throttles to OpenLCB, each of which may be in control of different trains on the OpenLCB bus.

Additional glossary and explanations:

- "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, typically using speed, direction and function commands. In addition to a model of a prototype train from locomotive to caboose, it might be just single locomotive, a caboose, a set of lit & controlled passenger cars, a diesel MU lash up, an operating crane or even a stationary object on the layout under control of the operator where the desired operator experience resembles how trains are controlled.
- "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, in 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. In this setup, OpenLCB acts as the throttle bus for the DCC Command Station.
- "Consisting": The running of multiple Trains 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~~Train Control Protocol, while leaving "configuration" to the configuration protocol(s). The line between these is admittedly

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

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.

2.4 Message Formats

2.4.1 Defined Event IDs

2.4.2 Defined Error Codes

~~Not a controller:~~ Permanent error – source not permitted – not a controller: 0x1021: For the use of this error code, see Section 2.6.1 "Controller".

Permanent error – not found: 0x1030~~Not found, already exists~~: For the use of this error code, see Section 2.6.5 "Listeners".

[2.4.3] ~~Traction~~Train Control Command Message

The MTI parameters are chosen to have the reply a higher priority than the request, ensuring replies to repeated instructions are always possible. The Type field expresses that these messages belong together. The Modifier field was chosen during the prototyping phase with a CAN-bus specific argumentation.

Dedicated OpenLCB messages are defined for ~~traction~~train control, instead of using datagrams, so that they have higher priority, and less complicated state machines. 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.

[2.4.4] ~~Traction~~Train Control Reply Message

This message is higher priority than the ~~Traction~~Train Control Command message to ensure that it can be sent immediately over ~~Traction~~Train Control Command messages. Coding and structure is intentionally 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 a bitset. 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:

191E9123 14 56 10 44 20 00 44 20

191E9123 24 56 44 20

which together are the single message

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

MTI=01E9 10 44 20 00 44 20 44 20

2.5 States

Speed states:

- It is recommended that Train Nodes exactly represent the 16-bit Set Speed in their internal state. There is no separate Direction state; the intended direction is determined by the sign bit of the Set Speed. Float16 can separately represent positive and negative zero. Implementations must take care that this distinction is correctly passed through all codepaths.
- Commanded Speed: This may lag behind the set speed intentionally due to momentum settings, etc. The Commanded Speed can not be directly set via OpenLCB; only indirectly influenced using the Set Speed and Emergency Stop commands. It is compliant to the Standard for an OpenLCB node to not report a separate Commanded Speed, in which case the Throttle Node shall assume the Commanded Speed to match the Set Speed.
- Current Speed. This state is optional, because not all OpenLCB Train Nodes are required to be equipped with electronic measurement of their physical speed.

The three state machines related to emergency stop need to be tracked separately by the Train Node. They act independently of each other. The Query Speed Reply flag related to E-Stop reports only the "Emergency Stop" state.

Function states: The non-specification means that different Train Nodes may have different number of functions, or no Functions at all. If they have only on/off functions, it is sufficient to remember one bit for the state, which is commonly represented as value == 0 and value == 1.

Listener states:

- Maintaining the order means that Listeners which were added earlier will be read out later as a smaller index in the Listener Query replies. Maintaining this order is necessary as some UIs may use this order to assemble a train (such as consisted locomotives) in the order stored here. To change the order, a Listener has to be detached and reattached, which brings it to the end.
- The recommendation about persistent state means that Listener configuration shall be retained across power cycles. This is a desirable product feature, where users are to assembling actual trains from multiple Train Nodes and then physically store these trains between operating sessions. The expectation is that next time they power up their layout, the trains are still in the assembled state that they were left.

2.6 Interactions

2.6.1 Controller

The purpose for maintaining the Controller in the Train Nodes is to support an operating model with control policies. The interactions are defined in a way to support different policies; the manufacturer of the Train Node can decide which one to implement or whether to make this configurable. Examples:

- Unique Controller. A train can be controlled from only a single throttle. In order for another throttle to select the same train, the train has to be "stolen" from the first throttle. The new

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

throttle and/or the old throttle may pop up messages to their respective user about the steal procedure; the message may be a notification only, or may even be a yes/no question to ask for permission to go ahead with the transfer of control.

- Shared Controls with Listeners. Multiple throttles can control the same train, and the Train Node will forward state changing requests between them. This allows all throttles to keep their display up-to-date with the current state of the Train Node. The last assigned throttle will receive heartbeats. The Train Node may still reject control messages if those come from a Node that is not a Listener.

- Unlimited. Any Node is allowed to send a Train Control Operation to the Train Node. This is helpful for example when automation software is controlling trains.

As the Controller is represented by the Node ID of the respective Throttle Node, another throttle can query the active Controller from the Train Node, and then reach out to the specific Throttle Node to determine metadata such as Simple Node Information to populate user interface elements. For example, a dialog may be presented saying "The Loco 'C&NW 415' is controlled by 'Joe's Throttle 123', would you like to steal it?".

The Controller is also used for Heartbeats, with the intention to prevent a runaway train. The requirement to detach the controller upon intentional shutdown is necessary to enable heartbeats. 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 operator will have to plug in again before the heartbeat interval expires.

A Throttle Node is recommended, but not required, to release the Controller position of a Train Node that is de-selected on the user interface. 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 sent an unsolicited message if it is currently assigned to a Train Node and another Throttle Node attempts to assign itself to the train as Controller. When the new Controller node requests to be assigned to the Train Node, one of several actions may occur.

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

2) The train forwards a Controller Changed Notify message to the currently assigned Controller node and that Controller node returns false. The Train Node returns false with a fail code of Controller Refused to the Assign Controller 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 returns 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 Controller request and the train node sets the new Controller as the active Controller.

The Release Controller Request does not have a reply defined. This request always succeeds.

2.6.2 Emergency Stop

When the Train Node goes into and out of Global Emergency Stop or Global Emergency Off, it will resume the previously set speed unless overridden during the Global Emergency Stop/Off state with a

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

different Set Speed command. This is necessary, because we want to stop all movement on the layout with these instructions, but we don't want to have to go through each train individually and set them to a new speed to resume operation of the layout.

If the Train Node is in Global Emergency Off/Stop state, any Set Speed commands will have no immediate effect and the train will remain stopped. However, the new Set Speed will be committed to the state, and when the Global Emergency Stop/Off is cleared, the train will accelerate to the new Set Speed.

However, when the train goes into and out of Emergency Stop state, the new speed will be determined by the Set Speed instruction that cleared the Emergency Stop state. For this reason there is no Clear Emergency Stop instruction.

2.6.3 Function Operation

For analog functions, it is not specified what the individual values mean or what is the acceptable scale. The Function D~~d~~efinition Information conveys the acceptable range to the Throttle.

It is not specified, how function values greater than 255 written using the Set Function instruction are represented in the Function memory space (0xF9), where there is only one byte available per function.

2.6.4 Train Identification

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.

Being a producer of the 'Is Train' Event (01.01.00.00.00.00.03.03) means that Train Nodes

- send a Producer Identified / Is Train message when they power up, and
- reply to requests for producers of that event.

An Identify Producers / Is Train request will therefore find all the Train Nodes on the OpenLCB bus, and further protocols can be used to get additional information on the individual Train Nodes it locates.

Throttle Nodes use:

- The Event Transport protocol to locate Train Nodes;
- PIP for enquiry about the support of ~~Traction~~Train Control Protocol;
- 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.
- Memory configuration, FDI, CDI & ACDI for getting specific information of a specific Train Node.

This protocol is only able to find Train Nodes that exist on the OpenLCB bus, and it can only find all of them in one go. Both of these are significant limitations: when a DCC Command Station represents DCC locomotives using Train Nodes, there are generally thousands of addresses that are not yet active, and do not have matching Train Nodes. Yet a user might want to select one of these addresses on their

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

throttle. On the other hand, a small and resource-constrained hand-held Throttle Node might not be able to deal with the responses from all Train Nodes (which could be hundreds) simultaneously.

An additional protocol, specified in the Train Search Protocol Standard is designed to cover these use-cases.

2.6.5 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.

There are two use-cases that are supported by the Listener functionality.

1. Multiple throttles driving the same train. The critical issue is that when one throttle sends a Train Control Operation to the train, the other throttle normally does not see that message on the OpenLCB bus. This is because the ~~Traction~~Train Control Request messages are addressed messages, i.e., point-to-point on the network. The solution is that both throttles registers themselves as a Listener on the Train Node. When Throttle A sends a Train Control Operation to the Train Node, that operation gets forwarded to Throttle B. This allows Throttle B to update its internal state and display with the effect of Throttle A's operation.

2. One Throttle Node driving multiple Train Nodes (also known as Consisting). In this case the different Train Nodes are connected to each other using bidirectional Listener links. The topology has to be loop-free. Any spanning tree is acceptable, where each link means the two respective nodes are registered as Listeners for each other. A star topology is the most efficient, but this is not prescribed. All links have to be bidirectional, because it is generally an important feature that the entire consist can be driven independently of which Train Node is dialed up on a locomotive.

For use-case 1 it is recommended to set the "Hide" flag in the Listener flag byte; for use-case 2 leave it clear. This allows consist manager software to be implemented and separate the two use-cases.

To discover and retrieve all listeners from a Train Node, multiple query messages are required. This presents a race condition when a third node changes the listener configuration during the process. It is not specified how the listener indexes change by attaching or detaching a listener. The querying node can recognize that their information is out of date by observing a change in the count of listeners compared to the expected number of listeners and restart the enumeration process.

A Train Node may provide additional configuration information, ~~for example as part of the~~perhaps with associated CDI, to customize the forwarding of messages. This may be interesting for Virtual Train Nodes specifically designed to be consist proxies.

A Train Node can distinguish on the received ~~Traction~~Train Control Request message whether it came from a throttle directly (P=0) or was forwarded from another Train Node (P=1). This is how the Train Node may and may not use this information:

- This distinction does not influence whether the message will be forwarded to any further listeners. Incoming Train Control Operations with P=1 are also forwarded; during forwarding they skip the link on which they came from.
- Set Speed and Emergency Stop message needs to be acted upon (and forwarded) by the train, independently of whether P=0 or P=1.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- Set Function message may or may not be acted upon for the local state if P=1, and the forwarding or not forwarding is defined by the Listener flags, independently of whether P=0 or P=1. The purpose of not acting upon forwarded function messages is that for Consisting, certain functions (such as horn/whistle) should only appear on the cab that the engineer is sitting in. Trailing consist members do not sound their horn. This is at the discretion of the Train Node, which typically means that this behavior may be configured in the Train Node's configuration settings.

2.6.6 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.

In case there is no assigned Controller node, there is no connection to check with heartbeats, and the train shall continue operating as last commanded. Controllers can choose to release themselves from the train if they wish to not receive heartbeats; this is recommended during any orderly shutdown of a controller or else the train will stop after the heartbeat period.

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

In addition to the memory spaces defined in this section, there are several other Memory Spaces defined by other standards that Train Nodes are recommended to implement:

- **Configuration** memory space.
  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.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- **Configuration Description Information (CDI)** memory space.
  The CDI space holds an XML description of the Configuration memory space. See the Configuration Description Information Standard.

2.7.1 Function Information 0xF9

Functions, such as lights and sounds, can be operated by the ~~Traction~~Train Control Set Function instruction, and their current value can be retrieved via the ~~Traction~~Train 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 ~~Traction~~Train Control instructions have two bytes per function value (support 0-65535), whereas the memory space only has one byte per function (supports 0-255). It is not specified how function values greater than 255 are represented in the Function Information memory space.

The NMRA S-9.2.1 describes DCC as having four separate sets of “functions”. The most common one is the traditional F0-F68. 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" accesses 127 addresses (the NMRA document implies that these are overlapping address spaces, but at least one manufacturer has not implemented them that way). Finally, it provides an “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 69+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. It is also acceptable to not store all of this information, as there is no requirement that a read shall return the last written value. For a DCC locomotive, it is recommended that the Command Station stores at least F0 to F28 and return their last written state upon a read.

2.7.2 Function Definition Information (FDI) 0xFA

So far there are defined ways to control a large potential set of numbered functions on a locomotive, but no information about whether these actually exist, what they do, or how should the user interact with them. The Function Definition Information is an XML file that the Train Node provides such that the Throttle Node can render a user interface that allows intuitive use of these functions.

The FDI is similar in intent and format to the Configuration Definition Information (CDI). It includes:

- Memory layout of the function values, defining what addresses control actually existing functions.
- Defining the data types and interaction modes, such as binary (on/off for lights), momentary (on while a button is held down, like horn), or integer values (e.g. volume).
- Function naming, so that a throttle can display useful names to the user such as "Bell", "Coupler Clank" and "Master Volume".

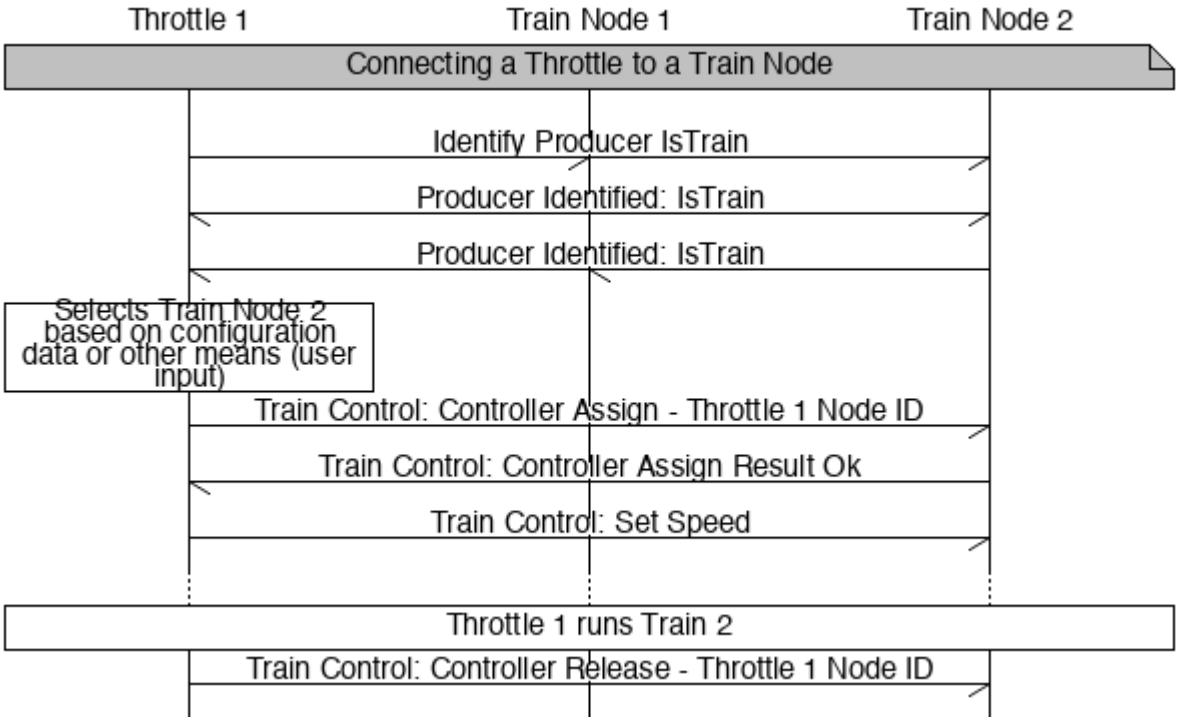
It is not specified in the Standard, where the information in the FDI should come from. It is up to the implementer of the Train Node. There are several different plausible sources:

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- A locomotive model with an OEM-installed OpenLCB decoder would have the functions determined and wired by the manufacturer. The manufacturer can then write the FDI as part of the product development and store it in the Train Node as read-only data, similarly to the CDI for any other OpenLCB Node.
- A generic or aftermarket OpenLCB locomotive decoder would have function mapping editable in the CDI, which it then uses to generate the FDI XML. This may also be generated and uploaded by an external tool, such as a sound project editor provided by the decoder manufacturer.
- A simple DCC Command Station may use a fixed FDI for all trains, which has just a generic description of F0-F28, or maybe encode conventions such as headlight, bell and horn.
- An advanced DCC Command Station may have a user-editable database (locomotive roster) about all DCC addresses, where the user can fill in information about the function mapping of each locomotive. This database can then be used to dynamically generate the FDI XML representation when a throttle is querying a given virtual Train Node belonging to a given DCC address.
- A Command Station with bidirectional communication to the locomotive decoders may use established methods for downloading the function information from the locomotive decoder and reformat this information into the FDI presentation. Example such protocols are RailCom Plus, RCN-218, NMRA S-9.2.1.1, mfx and M4.

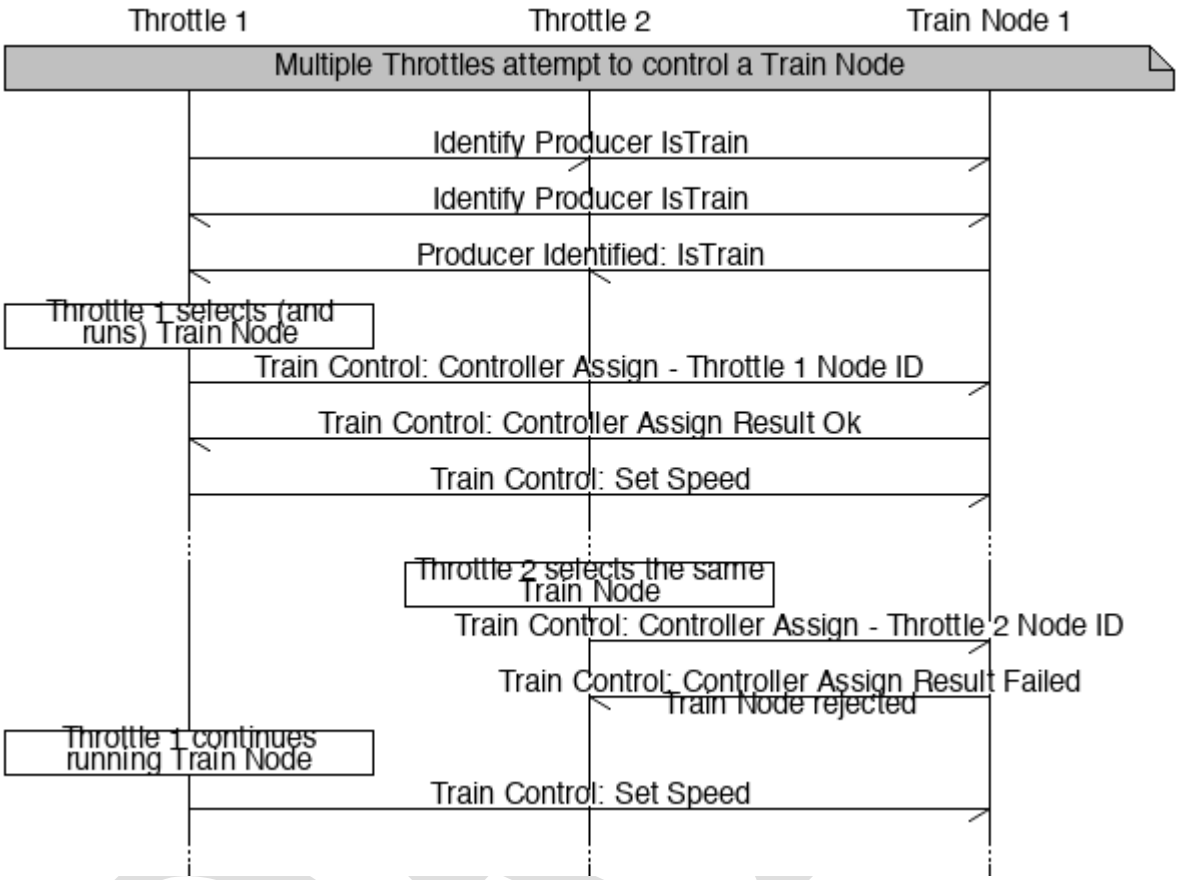
3 Examples

3.1 Basic Throttle and Train Connection



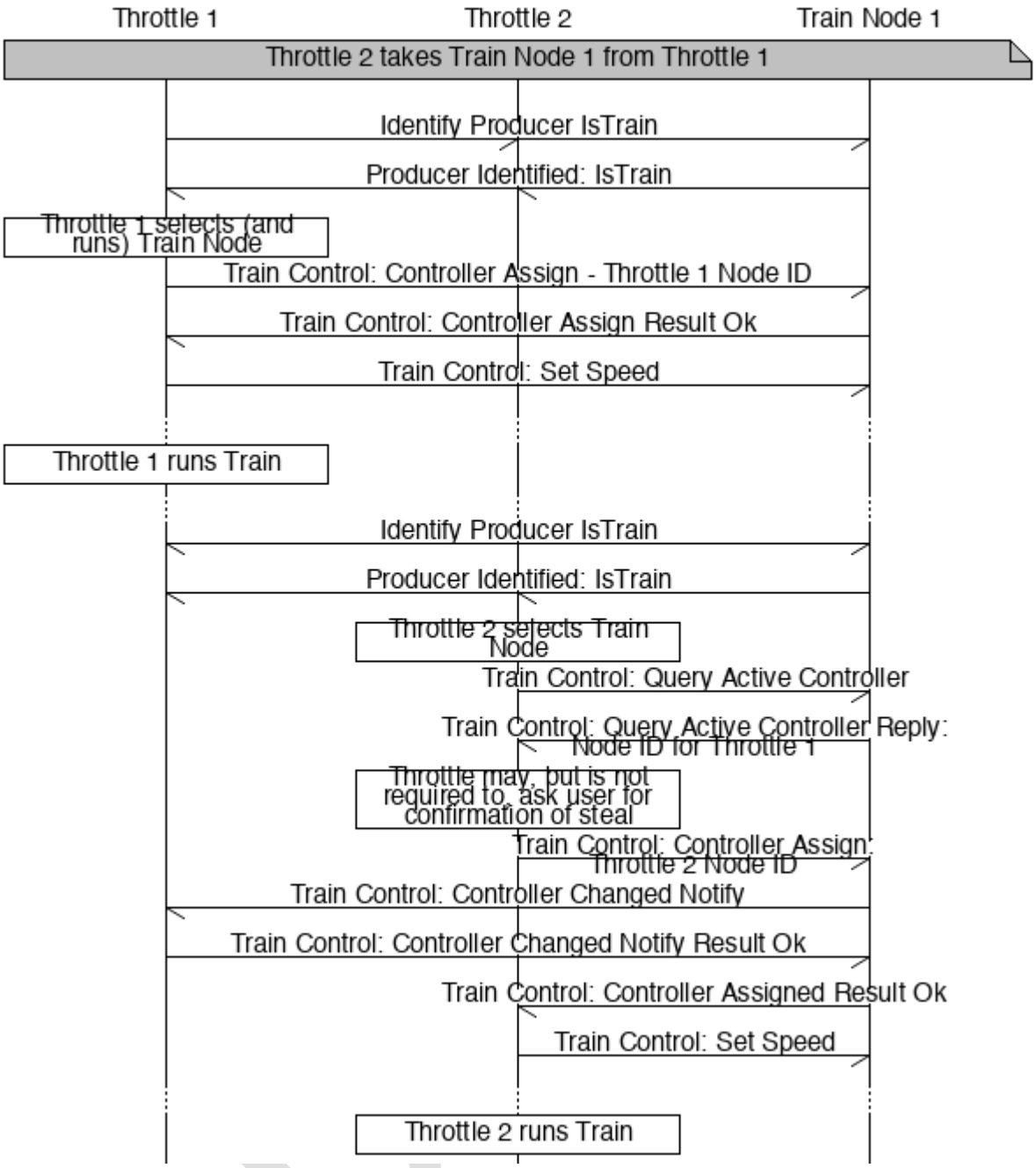
Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

3.2 Throttle to Throttle handover not supported



3.3 Throttle to Throttle Successful Hand-over (steal)

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.



4 Background Information

4.1 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 stop. The value specifies a speed in scale meters per second (scale-m/s). As an example,

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

he 16-bit quantity 0x5640 represents 100 scale-m/s, which is equivalent to ~223 scale-mph and 360 scale-km/h.

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.

A pragmatic way to deal with DCC-controlled locomotives without asking for further parameters is to set the top speed to 126 mph, and convert each speed step increment to 1 mph. Users interested in prototypical operations on DCC can acquire a speedometer and calibrate their locomotives’ speed table to 1 mph = 1 speed step, which achieves accurate speed display on the OpenLCB throttles. This display is then going to be accurate even if the throttle is configured to show km/h, without changing the DCC decoder speed table. DCC locomotives with lower maximum speed than 126 mph will have a flat section at the top of their speed table. Speeds above 126 mph will not be achievable using DCC, which is likely not a serious limitation, because model railroad layouts typically can’t operate safely at such high scale speeds.

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 throttle, at least without completely abandoning the particular interpretations assigned to speed 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

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

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.

4.2 Function Control

"Functions" like "horn", "headlight", etc. are key user features when operating modern decoders. From the protocol design perspective, they are very similar to configuration, in that:

- they are parameters that can be set remotely via the OpenLCB bus;
- these parameters effect the operation of the device;
- the parameters each come with a specific data type and accepted range or values (e.g. binary on/off or 0-255);
- we would like to make it easy for users to discover and understand what the parameters do (e.g. by providing a protocol with a user-visible text for each parameter).

OpenLCB has existing protocols for configuration that fulfill all of these requirements: the Memory Configuration Protocol that allows setting remote parameters, and the CDI description that covers the metadata requirements (both machine-readable and user-visible metadata).

Yet when we look at other protocols, such as DCC, as well as user expectations, functions and configuration are significantly differentiated:

- DCC has separate “functions” and “Configuration Variables” (CVs).
- CVs are persistent across power cycles, whereas functions are cleared upon a system reboot in many systems (although some do persist functions as well).
- Users want “Bell” to appear (automatically) on their throttle, but rarely want “Kp back-emf correction factor” to appear there. The expectation is that many functions are accessible with just a single button press.
- Functions are actuated significant more frequently that configuration. For this reason it is important that the protocol actuating functions is lightweight: should be low-latency, bandwidth conserving and should not take complex state machines.
- There are also parameters that are borderline between these two use-cases, such as “Master Volume”.

The design decision was to separate functions, while re-using patterns from the OpenLCB configuration mechanisms:

- A lightweight command is added to the ~~Traction~~Train Control Protocol to operate functions. It requires low bandwidth, and no state machines, handshakes or response message.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- Functions are a Memory Space, similar to Configuration, but at a separate space number. Memory Configuration Protocol can be used to bulk read or bulk write functions.
- For the user interface, an XML metadata description is specified just like CDI, called FDI. The XML metadata is downloaded using the Memory Configuration Protocol (just like for CDI). There is no binary protocol for the metadata, although a binary encoding of the XML contents were considered, but not (yet) developed.

The Set Function instruction is the light-weight mechanism for a controller to control the functions directly. The first argument is the address of the function to control as a 24-bit unsigned integer.

The ~~Traction~~Train Control Protocol does not define a semantics for function addresses; that is, there is no particular address that is singled out as representing headlights or the air horn. Such a mapping is thought to be too brittle, and there are just too many possibilities to be useful. Instead, the addresses are deliberately abstract, permitting the user to decide how to map addresses to functions for each train. This matches the DCC protocol's abstraction level; an address mapping is presented for DCC track protocol~~l~~. ~~,~~ M~~m~~ost other track protocols can also be packed relatively straightforwardly into the 24-bit address space.

The Set Function instruction takes 16-bit values. Most functions in current models are binary; thus 0x00 should be interpreted as "off", and any non-zero value as "on". Throttles are required to write "1" for "on" to reduce ambiguity. Analog (non-binary) FX should treat the 16-bit value as an unsigned integer.

Function values are stored in the 0xF9 memory space. There is only one byte available per function, and it is not specified how values above 255 should be handled. See Section 2.7.1 Function Information 0xF9.

"Function Definition Information", similar in intent to Configuration Definition Information (CDI) is stored in XML format in address space 0xFA to provide metadata for user interfaces See Section 2.7.2 Function Definition Information (FDI) 0xFA.

4.2.1 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. There is also a separate, independent numbering of electrical outputs by the NMRA and RCN standards that define decoder interfaces to the locomotives (FL, FR, AUX1, AUX2, etc.). These are often (and to much confusion for users) named as "function outputs" and the product is referenced in decoder manufacturers' marketing material with names such as an "8-output locomotive decoder"[1].

OpenLCB makes a clean separation between functions, which are the control operations, and all configuration & physical information, which lives in the configuration memory and CDI. If there's to be a mapping, it's defined through the CDI. The CDI contents are not standardized by OpenLCB, only the format. As such, there is no OpenLCB standard that defines what the "function mapping" is.

[1] Which, even more confusingly, means that FL, FR, and AUX1..AUX6 are available as electrical outputs.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

4.2.2 Legacy DCC Function Control

Although currently most DCC throttles only permit access to 13, 29 or 66 functions, DCC technically permits as many as 32961 different binary functions (see S-9.2.1: Function Group One Instruction, Function Group Two Instruction, and Feature Expansion Command Instruction Subcommand for F13...F68 Function Control, as well as Binary State Control short and long form). For this reason, it seems prudent to go ahead and use a 24-bit value for the address.

Likewise, although most DCC decoders only permit binary functions, some (for example SoundTraxx Tsunami sound decoders) also support a kind of analog control of effects 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. In recent years, the DCC standard was expanded by a canonical command of controlling analog values in the range of 0-255.

One problem that faces the decision to use a single command with a numerical function addressing system is that any kind of standardized assignment of function addresses (0, 1, 2, etc.) to particular effects (headlight, horn, ditch lights, etc.) is impossible in practice. DCC, for example, makes no such prescription, although by convention function F0 controls direction-sensitive headlights. Beyond this only manufacturers' conventions exist, which are not generally consistent across countries or continents:

- In the US, several throttle manufacturers mechanically label buttons with Headlight (F0), Bell (F1), Horn/Whistle (F2).
- In Europe, F4 by convention turns off momentum[2].
- In the US, F8 by convention *mutes* a sound decoder, whereas in Europe, F1 by convention *turns on* the sound of a sound decoder. Thus not only is the mapping different, but the functionality is also inverted.
- F2 is Horn/Whistle on sound decoders for both conventions.

As a conclusion, any kind of mapping is best handled on a per-train basis, by configuring the mapping between particular effects (e.g. headlights, air horn) to function addresses for each train, and OpenLCB leaves each address in the function address space uninterpreted.

Instead of functional mapping of addresses, OpenLCB maps them directly onto the addressing scheme used by the various control systems. 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 to perform the protocol translation without any user input of configuration. This however leaves users 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 function address space be mapped directly to the particular control system's address space; thus DCC F0 becomes function 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

[2] F4 is the highest addressable function number in the M/M II track protocol. Many early European command station & throttle devices thus had only F0..F4 as function buttons. Most locomotive decoders were based on an ASIC that had only AUX1 and AUX2 as electrical output, which were switched by F1 and F2, and momentum (and momentum off) were features implemented by this ASIC.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

beginning with 0b1.0000.0000.0000.0000), etc. For the complete mapping, see the ~~Traction~~Train Control Protocol Standard, Section “Function Information 0xF9”.

Other systems should be handled analogously. The expectation is that without any metadata specifically configured by the OEM or the user for the given locomotive, whenever a the user presses buttons 0..9 on an OpenLCB throttle, the same track output is generated as if a native throttle of the given system would press the respective buttons 0..9, if applicable.

4.3 Configuration and Metadata

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

4.3.1 Roster Information

The configuration information in a train can include the user documentation and editable fields that are sometimes referred to as “roster information”. This might include owner name, prototype railroad and road number, information about the particular model's construction, etc. An important user journey is to define identification information for a locomotive. See Section 4.3.2 (Train Identification) on how this is expected to be used.

Future work:

Possible option to standardize this area would be to extend ACDI/SNIP, or define a similar protocol. ACDI/SNIP currently have a block for manufacturer identification, and a block for user identification. Those are both versioned. The following approaches were proposed, and should be considered as future work:

- Extend ACDI/SNIP 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 new version of the user block, creating additional fields, specific to Roster Information.
- Define a new memory space for Train Roster Information (with a fixed layout modeled from ACDI) and a new message to request/provide this as Simple Train Node Ident Info Protocol. The MTIs 0xDA8/0x9C8 were tentatively reserved for this use-case, but at the time of this writing, no open-source prototype implementation or system in production uses this.

Suitable content for a (first version of) this might be (from JMRI roster[3], see also similar concepts in [rocrail.net RocRail]):

- Road Name
- Road Number

Manufacturer, model, owner description, comments, etc are already present in ACDI/SNIP.

For DCC locomotives, more terms might be desired:

- DCC Address

[3] https://www.jmri.org/JavaDoc/doc/jmri/jmrit/roster/RosterEntry.html

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- Decoder Type (Manufacturer, model)

4.3.2 Train Identification

The most important metadata is the piece of data that is used to identify a train. For the OpenLCB bus, this is the Node ID, but it is not intended to be a central piece of user interfaces. There are two critical user journeys where identification is needed:

- How does the user select a locomotive when they have a throttle in hand.
- When a throttle has a locomotive selected, how is that displayed on the screen.

As convention, OpenLCB throttles are expected to use the "User Name" field (from SNIP and ACDI) as locomotive name on the display. The Train Search Protocol specifies that the User Name field shall be searched for cab numbers when a throttle is searching for or attempting to select a locomotive.

This means that users are expected to put into the User Name field what they would like to see on their throttle, typically the most important identifying information, such as road name (e.g. "C&NW"), cab number (e.g. "415"), and maybe the prototype model/type (e.g. "F7A"), depending on how long of a name their preferred throttle can display.

Background information:

Historically, digital command systems used a unique locomotive address as identification. The user would set the locomotive address using permanent configuration, is expected to remember what address was set for a given locomotive, and type in the locomotive address on a numeric keypad to select a locomotive.

For the combination of typical US railroads and DCC protocol, this mechanism works very well due to several specific reasons:

- DCC supports locomotive addresses up to 4 digits.
- Most US railroads use 2 to 4 digit cab numbers.
- Cab numbers are printed with large letters on a side of a US locomotive, making it easy to read on a scale model.

This established the procedure of "take the cab number from the loco, punch it in and drive".

However, none of these hold as general assumptions across the world and across digital command systems:

- Motorola trinary protocol supports 80 addresses (255 in an extended interpretation); Selectrix supports 112 addresses. This is not sufficient to cover even US cab numbers with a 1-to-1 mapping.
- The mfx/M4 track protocol does not have persistent addresses that the user would assign to a locomotive. Instead, decoders are uniquely identified with a 32-bit number, and the Command Station provides a session address which is 11, 14 or 28 bits long. The user is not exposed to either of these address schemes. The RCN-218 and NMRA S-9.2.1.1 standards are defining a similar scheme for DCC, where the user would not need to manually assign unique addresses. This also decouples the address on the track protocol from any potential cab number.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- Rolling stock in Europe 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 (road), the type of vehicle and a unique number in its series. It might look like "91 85 0 474 014 2". Even with the most reasonable shortening, this collapses to "474 014", which is too long for a locomotive address for all track protocols mentioned here.
- The UIC number is printed on rolling stock using about 6" letters. Half a scale foot is very difficult to read from models in most scales.

As a conclusion, inheriting the identification mechanism from track protocols such as DCC does not fulfill the requirements on general use-cases. On the other hand, the User Name in SNIP/ACDI is defined as up to 63 byte long UTF-8 string, which comfortably covers the necessary use-cases.

Future work:

Modern command stations & ecosystems often contain high performance input devices with high-resolution color touchscreens. A reasonable expectation is for locomotives to be represented by images rather than text. This is true both for selecting a locomotive (a user sees a scrolling list with images from all of their locomotives), as well as while driving one or more locomotives (the currently selected locomotive image is visible on the screen).

There are three different conceptual sources of the image data:

- a central image database created by the command system manufacturer, or a cross-manufacturer organization (including OpenLCB, NMRA or RailCommunity);
- an image or drawing created by the locomotive OEM[4];
- an photograph uploaded by the user.

An upcoming standard RCN-218 / TN-218 and and NMRA S-9.1.2.2 provide three pieces of metadata to represent locomotive imagery:

- A 2-bit locomotive category (Steam, Diesel, Electric, Railcar).
- A 16-bit locomotive enumeration, which can index into a database in the command system, intended to contain drawings or silhouettes of all types covered by large runs of major manufacturers.
- A string intended to contain a URL.

In addition to these, a command system can use the unique identifier of the train to key into an internal database of image data uploaded by the user. Both RCN-218/S-9.2.1.1 as well as OpenLCB have unique hardware identifiers suitable for this purpose. These two are not linked together however.

4.3.3 Legacy DCC Configuration and CVs

Representing CVs of a DCC decoder is necessary when an OpenLCB-to-DCC gateway is in operation. In this case the Train Node is presented for the OpenLCB bus by the gateway, and other OpenLCB Nodes are attempting to perform CV configuration of the connected DCC decoder.

The Gateway may be

- A DCC Command Station, representing many DCC locomotives using DCC program-on-main commands and possibly RailCom read-backs;
- A DCC Programming Track, which may or may not be built into a DCC Command Station;

[4] OpenLCB has enough bandwidth to download this information directly from the Train Node, provided that the system has an effective caching mechanism.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- A wireless OpenLCB Train Node, built into a locomotive, which generates a DCC output signal for a single standard DCC decoder, which then in turn controls motor, light and sound functions. This is a common setup for battery-powered locomotives / command systems.

A DCC Programming Track is not a Train Node. It does not generally implement the ~~Traction~~Train Control Protocol, because none of the Train Control Operations are available on the Programming Track[5]. There has to be a mechanism to find Programming Track nodes on the OpenLCB bus; a production system accomplishes this using the Event Protocol with Identify Producer / Producer Identified messages for a specific event ID 09.00.99.FE.FF.FF.00.02.

The intention is to provide a protocol that both the Virtual Train Nodes as well as DCC Programming Track nodes can implement for manipulating DCC CVs. Since there are no low-latency or high-bandwidth requirements on CV reads and writes, the Memory Configuration Protocol is a well-suited transport mechanism. A production system uses memory space 0xF8 for this purpose.

There are two different use-cases of how client Nodes want to operate this protocol:

- DCC-specific CV programming protocol. DCC Throttles (or other user interfaces) provide a generic CV programming experience, where the user punches in a CV number, a value and presses Program button. Computer programs such as DecoderPro provide APIs for CV programming and rich parametrized UIs that know details about individual decoders. The OpenLCB Programming Track / Train Node does not need to know about the semantics of the individual CVs of the given decoder.

- An OpenLCB Configuration Tool, which is unaware of DCC specifics, and given an appropriate CDI definition is possibly able to perform the necessary Memory Configuration Protocol writes/reads to the Memory Space occupied by the DCC CVs. The CDI in this case would typically have to be specific to the particular DCC decoder, or cover only the standardized CV set.

The aforementioned production system uses the following layout for the DCC CV Memory Space:

- Address 0-1023 is read/write (byte-oriented) access to CV 1-1024. (Note: On the DCC track protocol these are actually indexed as 0-1023.)

- The DCC Train Node defaults to POM writes / RailCom reads, the Programming Track defaults to DCC Direct Mode writes/reads.

- Address bits masked by 0xFF000000 are modifier bits as follows:
  - 0x00: use default programming mode as above, byte-oriented.
  - 0x01: use direct mode programming (only valid on the Program Track), byte-oriented.
  - 0x02: use program-on-the-main, byte-oriented.
  - 0x03: use paged mode programming (only valid on the Program Track), byte-oriented.
  - 0x10 – 0x17: use default programming mode, bit oriented, masking bit 0..7 respectively.

[5] If the Programming Track is to be used as a regular DCC track (mainline), then it is the Virtual Train Nodes that represent the regular DCC addresses that implement the Train Control Operations and thus the ~~Traction~~Train Control Protocol.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

Future work:

The above statements need to be moved to the standard in a future revision, as-is or revised.

Significant past consideration has been given to certain special cases that happen in DCC manufacturers' CV interpretations, with the aim of providing a mapping of the Memory Space that would allow a CDI XML be written that represents the decoder features accurately and correctly.

- Indexed CVs. Example by QSI: write 12 to CV 51, then 8 to CV52, and then CV54 is the value to read/write. Example by TCS: write index-hi to CV 201, index-low to CV 202, then CV 203 and CV 204 is a 16-bit value to read/write. Example by NMRA: write index-hi to CV31, index-lo to CV32, then read/write CV 256 to 511 to access a set of 16 million CVs. A proposal to consider is to use a 3-byte address to represent the CV number, with the high two bytes representing the index values. The MSB (fourth byte) has to encode which indexing method to use, if more than one is to be supported.
- Bit-masked CVs such as CV 29 or CV 28 can be handled with a bit selector as above.
- For arbitrary length bit fields, a different bit selector mechanism was proposed: 0xFF 00 53 3C would access the middle bits (the 0x3C mask) of CV 53, and the 0xFF is just an arbitrary key for representing this. The mask can not span across a byte boundary.
- Multi-byte writes such as CV17/18 can be performed with a single Memory Configuration Protocol datagram.
- Writes to CV1/CV29 need a special sequencing; the proposal was to ignore this entirely at the CDI level & build it into the gateway to DCC.

It's important to note that none of these advanced considerations are necessary to make a software like DecoderPro work; DecoderPro does not rely on CDI-compatible features. This also means that there is limited returns on defining and implementing complex logic for weird cases that DCC decoders exhibit, because DecoderPro actually covers them.

4.4 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 Train Control Operations 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 Set Speed commands, which will cause the consist to move together.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- It is possible to control a single Train Node from multiple throttles. Each throttle has to register themselves as a Listener on the Train Node. Then when one throttle changes the locomotive state, all other throttles get a copy of the message and have the ability to update their displays real-time. This gives a seamless experience for multiple operators when using encoder-based throttles.
- Throttles that relinquish the control of a locomotive (through stealing/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.
- A supervisor software can observe the operator driving a train and intervene when certain policies are not complied with (e.g. if the train runs a signal).

How to handle consist topologies:

When creating or managing a consist, the listener ~~relation~~relationship is recommended to be established in a symmetric manner, where each link in the consist is bidirectional. This means that if engine A and B have a link, then B is a Listener in A, and A is a Listener in B. Symmetric listener relations are allowed and cause no problem for the network~~,~~ because a Train Node is required to never echo a message back 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. If the tail engine is backwards in the consist, then driving the consist from that engine also changes the relative direction that the throttle sees. This is prototypical, it means that after a runaround move and selecting the other cab, the train is driving in “forward” state of the reverser.

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.

When more than two locomotives are part of a consist, there are multiple choices of how to arrange the (bidirectional) links with regards to each other, as there are multiple possible spanning trees across the Train Nodes. From the perspective of the OpenLCB protocol, any spanning tree will work correctly and the states (and state changes) of the locomotives will be indistinguishable. Example topologies would be a “star” topology where one Train Node is selected as “lead” and all others are consisted with a single link to the lead; or a linear topology where every locomotive is consisted to the previous and next according to a given order. The star topology minimizes the latency of all locomotives receiving a command. In the presence of both wired and wireless locomotives, it is not recommended to select a wireless locomotive as the center node of a star.

It is not recommended to create unidirectional consist links. Consist management software may automatically, or upon request by the user, fix consists that have a problem with their links such as a unidirectional consist link, or rearrange the topology of the links according to some policy.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

If an application requires representing and storing the order of members in a consist, a star topology allows to do this. OpenLCB nodes are required to retain the ordering of the registered Listener links. This means that the center of the star would be the first locomotive, and all the remaining locomotives are in the order as registered at the first locomotive. There is no operation to change the order of listeners, consist manager software may have to delete and re-create multiple links in the desired order to change the stored order. It is not possible to create an order between two locomotives using this technique.

How to handle functions in consists:

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

- Forwarding speed commands only, no functions. (It is possible to select inverted direction, when the following locomotive is backwards-facing in the consist).
- Forwarding function zero (usually light) commands only. 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.
- Forwarding all function commands~~s~~. Useful for the throttle listener use-case or in case the two locomotives are equipped with identical decoders and features.

This provides several possible mechanisms for configuring function forwarding in the consist:

1. The simple model, where only the two fixed configuration bits are available for each consist member. This is suitable for throttles that do not have the capacity to present arbitrary configuration.
2. In an advanced configuration each Train Node is responsible for making its own interpretation of the received function messages. The Train Node can use its configuration and the incoming message's bit P to determine whether a function should be applied locally or just forwarded. In this model it's essential that the Listener links are all set up with both F0 and Fn forwarding enabled.
3. In case of the unidirectional topology using a central Node, that consist leader has to have sufficiently complex configuration to be able to determine which function messages to forward where.

Operations:

Once the consisting relationship has been set up, it will handle the individual parts of the ~~Traction~~Train Control 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.

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

- 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 Search and Controller Assign – Each member in the consist, as well as a Virtual Train Node, if used, can be separately located by the standard protocol. They will take part in producing the well-known events, providing human readable (and writable) information via SNII/ACDI, and being the target of search operations. If the consist has unidirectional links, it can be operated only by the lead engine. This also means that the assigned Controller is meaningful on the lead Train Node. Consists with symmetric listener relationships can be driven by calling up any single node in the consist. This means that there can be multiple assigned Controllers at the same time; throttles are recommended to use the Listener functionality themselves.

Because there's no protocol difference between full-OpenLCB-node Trains and legacy equipment that uses an OpenLCB gateway via a Virtual Train Node, 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. A Gateway can handle multiple node IDs (and/or DCC addresses), which allows implementation of consisting. Speed/direction just passes through; speed matching is automatic, provided that the locomotives were mechanically calibrated to the correct scale speeds.

DRAFT

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

Table of Contents

DRAFT

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.

DRAFT

Copyright 2011-2024. All rights reserved. This OpenLCB document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). See https://openlcb.org/licensing for more information.