OpenLCB - *A new way of controlling model railroad*

Use Cases

This document discusses how the OpenLCB™ meets various use cases. OpenLCB has been developed by the OpenLCB group via their web site at http://openlcb.org/ and discussion group at http://groups.yahoo.com/group/OpenLCB.

Each section describes a single use case, and then discusses the OpenLCB solution for that case.

Entry Level User

A model railroader wants to learn about OpenLCB. He buys two inexpensive boards at his local hobby shop. He connects them on his layout to two pushbuttons and a turnout motor, does a simple configuration, and is able to control the turnout with the pushbutton.

Solution

Alice wants to control her three track station. She has goes to her local store and buys one 8-button input node, four turnout-controller nodes, and a power supply. When she gets home, she unpacks the nodes and connects each of the turnout-controllers to her four turnouts, E1, E2, W1, and W2. Then she connects the nodes together with the provided cables. Following the directions, she then uses the blue button on each of the turnout-controllers and ensures the attached turnout changes position.

Now she is ready to make the buttons control the turnouts. The buttons are labelled A-H. To connect A to turnout E1, she pushes the blue button on the E1 controller until the turnout is in the position she wants, and then the gold button – this tells the controller to learn this position. She then goes back to the button node and pushes the gold button, button A, and then the gold button again. This teaches the turnout controller to respond to button A. She then sets E1 to the other position and trains it with button B. (In more detail, on E1 she pushed blue repeatedly to position the turnout, then pressed gold, then pressed gold, B, gold on the button node). She repeats this with turnouts E2, W1, and W2 and C-H.

She finds that she occasionally forgets to compatibly position the turnouts at both ends of the station. However, she realizes that she can improve her scheme. She resets the nodes to factory settings as directed and then teaches button A to make all the turnouts select track 1. (In detail she: on E1 sets diverge and gold, on E2 sets diverge and gold, on W1 sets diverge and gold, on W2 sets diverge and gold, and finally on the button node pushes gold, A, and gold). She also makes buttons B and C select tracks 2 and 3.

Now she can control her station tracks with only three buttons, and there is no chance that they can be misaligned! Later, she changes it again so that A-C selected tracks 1-2 for entrance, and D-F selected them for exit.

Expanding a Small Layout

A user has four OpenLCB boards working on his layout. He buys one more input board to provide extra pushbuttons to operate his yard ladder from the other end of the layout. He's able to configure the new board and put it into operation.

Solution

Alice decides that it would be nice to be able to also control her station tracks from the other side of her peninsula. She buys another 8-button node (lets label these with M-T), and hooks it to one of the other nodes. She then copies the learned event from the first button node to this new one. (In detail: on the new set of buttons she pushes blue, M and gold, and then on the original button-node she pushes gold, A and gold, and repeats the same process with the other buttons).

Mid-size Layout Problem

A user has twelve OpenLCB boards operating. Something changes, and the system is no longer reliable. Certain operations no longer work, and others only work sometimes. The user attempts some debugging steps using features of the existing hardware, but is unable to locate the problem, so buys/borrows diagnostic device(s).

Solution

Mike is at a loss, although he kept good records things have gone south on his layout. He calls up his friend and borrows his laptop with the OpenLCB configuration software and a OpenLCB-USB adapter. He plugs them into his layout LCB and turns it on. He asks the program to interrogate an unknown layout, and the program lists his twelve boards. He drills down to each part of each node and sees what other nodes receive its events. He also sees whose events it receives. The program also draws a diagram of the nodes. He pushes some buttons and watches as the program animates the events send from one node and to the receiving nodes. He finds his mistake, and makes changes to his configuration and tests again. Problem solved – he vows to buy a laptop.

Prototypical Signals

Jim wants to add signals to his layout, and wants them to be as prototypically correct as possible. He has sources for his preferred signals, block detectors, switch motors, etc and wants to use his own choices. He wants this to work with both a physics (switches and lights) and glass (computer screen) dispatcher panel.

Solution

Jim buys suitable OpenLCB boards to connect up all his components. These boards will be of several types, perhaps from several vendors, but all coexist without conflicts on his OpenLCB segment(s). He also obtains (buys or writes) software for his computer that implements the specific signaling rules for his prototype. Either that same software or another package will also provide his dispatcher panel logic. He hooks his computer to the OpenLCB segment(s) with a USB or Ethernet bridge. He then teaches the software about each of his input and output connections. This can be done with programming buttons, or by entering node information directly into the program. After the initial setup is complete, he can then work with the software to make it do exactly what he expects from his prototype.

Large Layout Upgrade

A model railroad has 40 OpenLCB boards controlling a large layout. To upgrade a major yard, the user wants to replace 12 boards with new ones that offer new (non-OpenLCB) features. He captures the existing setup, replaces old boards with new ones, configures the equipment and is back in operation.

Solution

Joe has been assigned the non-trivial task of upgrading the yard with the new boards the planning committee bought. He tells the OpenLCB configuration program to make a fixed record of the configuration of the twelve boards he is replacing, and saves it to disk, twice, he's paranoid. He then proceeds to disconnect the original twelve boards and replace them with the new ones. Sometime later ... he brings up the configuration program, and its informs him there are twelve new boards on the layout. “Tell me something I don't know” he mutters. He brings up another copy and retrieves his stored configuration. He notes that some of the names of things have changed (new and improved!), but he can manage. He uses the old configuration to find nodes from which he can teach the old events to his new nodes – this is easy, on the screen he just grabs the node's consumer or producer entry and drags it to one or more of the new nodes' consumers and/or producers. The nodes are linked and using the old events. “Sweet” he thinks. Now he sets about to use all those new capabilities that these new nodes have – “Wow, it was worth it, these babies are hot”.

Distant Control Panel

A model railroad has turnouts and signals controlled via OpenLCB. The owner wants to put a physical control panel in his house, a separate building, and operate it via OpenLCB.

Solution

Mike has been operating his layout from afar using a 'glass' control panel on his computer connected to his layout via his Wifi router. However, it doesn't have the same satisfaction as having a real control panel. He decides to build one, and he installs a number of CAN OpenLCB nodes to interface it. The big day arrives and now he needs to configure it. He connects a CAN-to-Ethernet bridge and plugs it into his networks. He boots up his configuration program, links to the bridge and sets it up to talk to his layout. He then links to his layout, and the program announces there are lots of new nodes. Using drag-and-drop he sets his new nodes use the the events of the existing layout. This is relatively easy, as he is just duplicating his old controls. He can just drag events from an old node onto a producer or consumer in the new one to configure it. The whole process takes only 15 minutes. Mike thinks “Now, let's get down to some serious operating”.

Large Layout Expands

A large model using CAN, but not any attached computers, grows enough that it needs to have another CAN segment. This must be possible without reconfiguring all the existing nodes, event IDs, etc.

Solution

Hugh realizes he has a problem – the OpenLCB-CAN bus running his layout is not big enough, he's maxed out the number of nodes. He talks to the local guru, and the guru says – no problem, just divide your CAN into two parts and join the parts with a CAN-to-CAN bridge. Hugh is doubtful, it can't be

that simple, won't he need to change node ids and change his event programming? The guru reassures him – it is that simple – no need to change anything, the nodes will still see all the other nodes events, just as before. Hugh takes him at his word, picks up a bridge and, after five minutes deciding where to break the CAN bus, inserts the bridge and is running trains again.

Connect Multiple Programs

The user wants to run multiple programs from multiple vendors in his home computer that simultaneously connect to the layout OpenLCB and control/monitor it.

Solution

Mike visits Hugh and brings some new OpenLCB software on his laptop. “This software will let us make a mock-up of your layout showing the location and connections between your nodes. I just need to plug it into your layout”. Hugh copies it to his laptop, and plugs it into the opposite end of the layout – instant operation from both computers.

Remote Dispatcher

A model railroad has turnouts and signals controlled via OpenLCB. OpenLCB is used to install a CTC panel at a distant location. The panel could be either a physical panel or on a computer screen.

Solution

See Distant Control Panel. The same solution that works over a local network will also work over both high-speed and low-speed connections to the global Internet, and thus to where-ever the dispatcher is located.

Modular Layouts

A modular club has fifty modules, each of which as a CAN OpenLCB with two or three nodes controlling the module. These modules are separately built, with no central administration. For example, a new member might be bringing four modules that have never run with the club before, or a member may have just completed a couple new modules. They are brought to a central location for a meet, where they are all connected together in some pre-planned orientation. The OpenLCBs are connected in some fashion, and used to operate the entire layout from both central and distributed locations.

The club does not want to expend any effort preallocating numbers, making sure that module configurations don't conflict or anything like that; they just want to plug together the OpenLCB CAN backbone and operate the entire layout from both central and distributed locations. All the modules must still continue to operate without reprogramming or reconfiguration; there's just not enough time for that. Connections between modules need to be added easily, either by multiple manual operations (pushing local buttons) taking place in parallel, or multiple computers attached to the layout with configuration tools.

Solution

John is in charge of the meet. He tells the attendees to connect their modules together into multiple CAN segments, observing the limitations of length and drive capabilities of the nodes. He assigns segment leaders and instructs them to configure and debug each segment in isolation by connecting a computer with configuration software. XML files describing the modules that some attendees brought help in this, but the nodes themselves contain enough information to allow them to be connected and integrate their existing block detection and signaling systems.

If the number of modules was smaller, then John would have connected the segments together using CAN-to-CAN relay nodes, but he decides to connect the segments together using an Ethernet backbone, using Ethernet-to-CAN bridges. This will reduce the bandwidth requirements on each of the segments because the bridges will automatically only pass events between the segments that are actually required, rather than all packets. Working with his segment leaders, they quickly integrate the segments onto the backbone. Since they have multiple panels, both real and virtual, attendees can observe and control the layout from disparate points on the layout.

Remote Diagnostics

A club layout is operated via OpenLCB. There's something not quite right about the signaling, and one of the members wants to check the operation of the signals. He makes a remote connection to the layout and checks the status and configuration of the hardware from his home computer.

Solution

Bill receives a call from his friend Hugh that there's some trouble on Hugh's layout. Bill is an expert on model railroad signaling, but he lives half-way across the country from Hugh's layout. Fortunately Hugh has an Ethernet connection to the layout, and even better he has attached tit o the local area network (LAN) and to the wider internet through a router – Bill can just log in from where he is! He opens his copy of the configuration program on his home computer and connects to the layout. It appears that one of the occupancy detectors is not seeing the train, and this has lead to a cascade of errors downstream from the signal. He phones Hugh back and tells him to check the connections on that block. Hugh crawls under the layout and discovers a plug has pulled out – it must have been accidently hooked out by someone in the work party mid-week. The plug is replaced and everything returns to normal operation. OpenLCB

Aggregation of Modular Clubs

Dozens of clubs put together dozens of OpenLCB segments and hundreds of modules that have been separately configured. A large FREMO meet would be an example. Collision avoidance must be easily arranged; event and node identifications that are already configured into the modules should already be unique, without any need to reprogram ones that might be duplicates. It must be possible to build automated tools for connecting events between layouts and modules. Conventions and/or tools for connecting across the boundaries are needed so that e.g. signaling systems can work with adjacent modules they've never met before. It must be possible to build automated tools for health monitoring across the layout.

Solution

See Modular Layouts. In this case, thanks to prior planning, each club has provided an XML file describing the events that must be connected to allow the integration of the signaling and other systems. Hugh and John use the configuration program to build a schematic diagram of the entire layout. They then drill down to each interface between layout segments and connect the edge nodes of each segment together. They quite quickly repeat the exercise on each adjacent pair of segments, with the help of the leads of the pairs of segments. Since every node has a unique ID and unique Events, they are not worried about conflicts or having to change ranges of events on any modules – reprogramming hundreds of nodes would not be inviting.

Table of Contents

